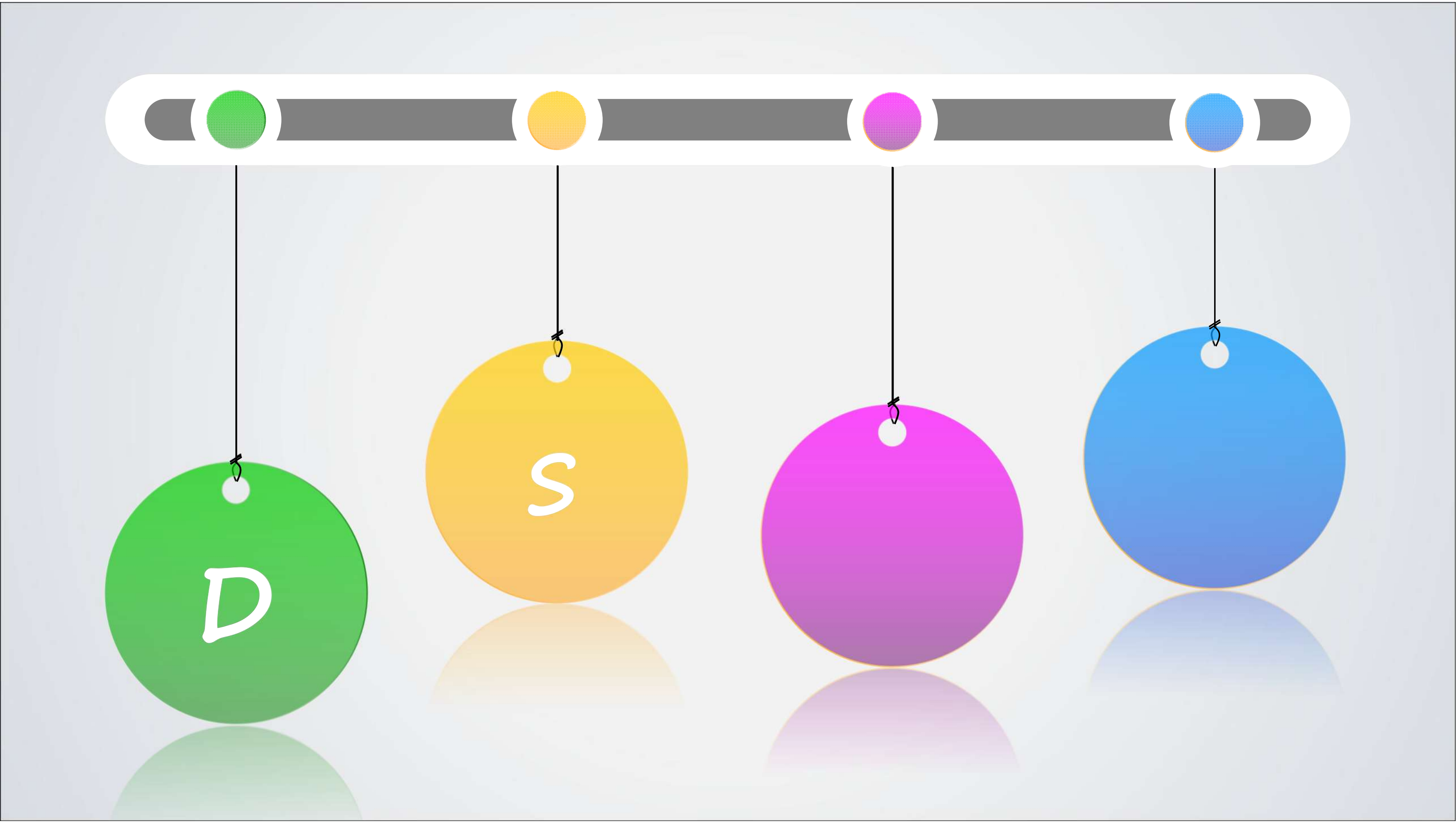

D
S

1

Trees:
Basic Terminology of Trees

2

- Representation of Binary Tree
- Tree Traversal Techniques-
  Pre order, In order and Post order

3

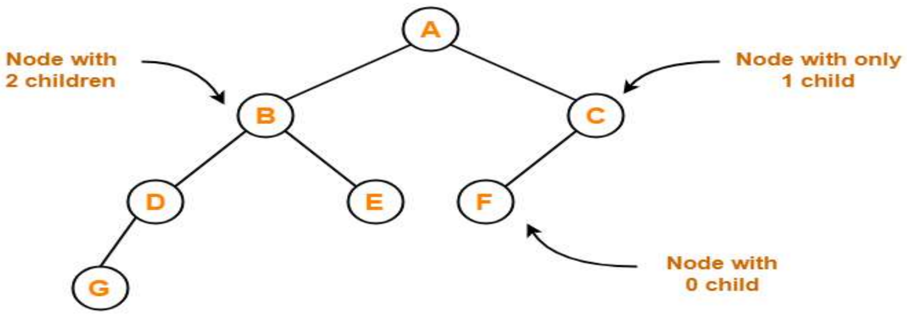
Introduction to Graphs,
Graph Terminology, Types

4

Representation of Graphs

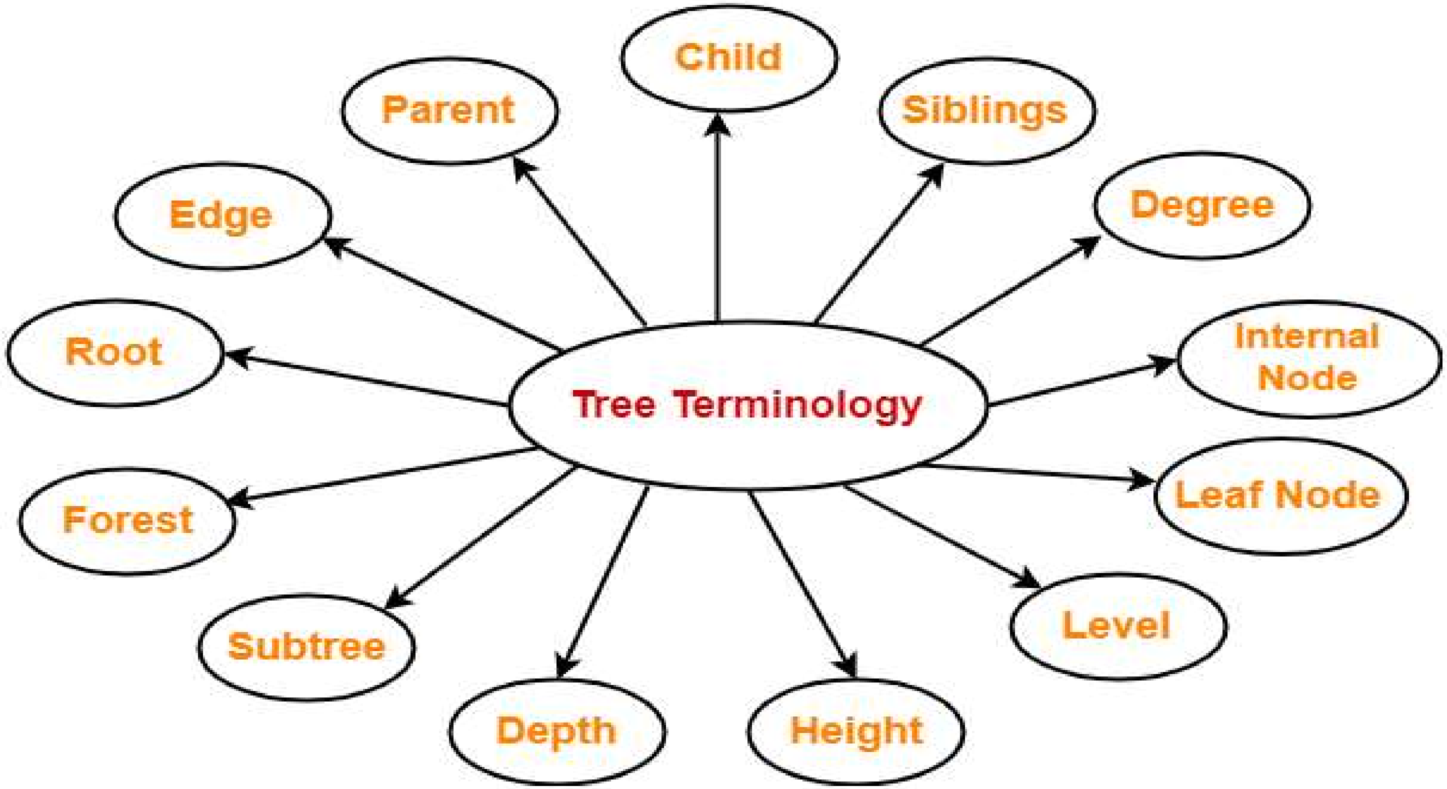
Definition of Tree

- A tree is a **non-linear data structure made up** of nodes or **vertices and edges without having any cycle**.
- The **tree with no nodes** is called the null or **empty tree.**
- Tree organizes data in **a hierarchical structure**
- **In a tree** data structure, a **node can have any number of child nodes.**
- **Binary tree** is a special tree data structure in **which each node can have at most 2 children.**
- Thus, in a **binary tree, Each node has either 0 child or 1 child or 2 children.**



Binary Tree Example

Basic Tree Terminologies



Basic Tree Terminologies

1. Root:

- The **first node** from **where the tree originates** is called as a **root node.**
- In any tree, there must be **only one root node.**

2. Edge:

- The connecting **link between any two nodes** is called as an **edge**.
- In a tree with **n number of nodes**, there are exactly **(n-1) number of edges**.

3.Parent:

- **The node which has one or more children** is called as a parent node.
- In a tree, a parent node can have any number of child nodes.

4.Child

- The node which is a **descendant of some node** is called as a **child node**.

Basic Tree Terminologies

5. Siblings:

- Nodes which belong to the **same parent** are called as **siblings**.
- In other words, nodes with the same parent are sibling nodes.

6. Degree:

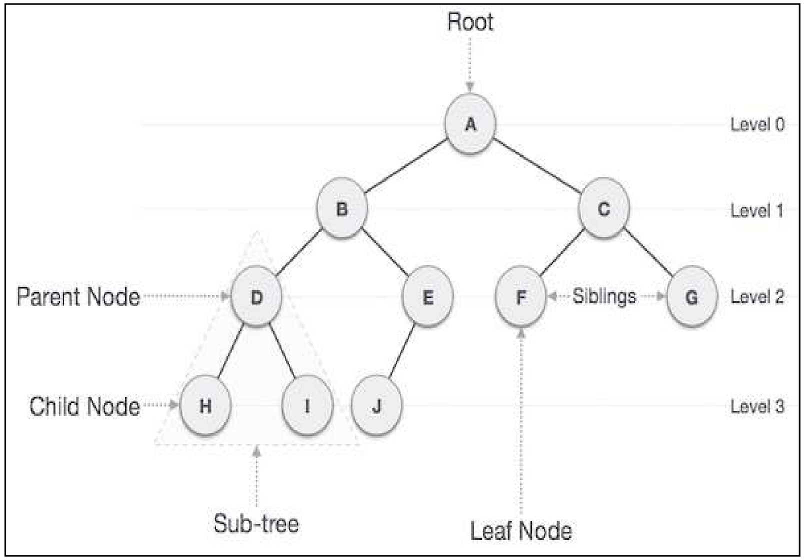
- **Degree of a node** is the **total number of children of that node.**
- **Degree of a tree** is **the highest degree of a node** among all the nodes in the tree.

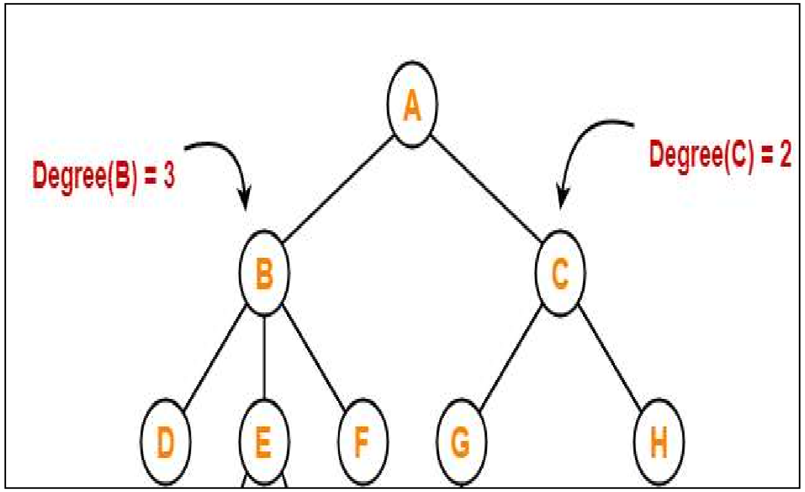
7. Leaf Node:

- The node **which does not have any child** is called as a **leaf node**.
- Leaf nodes are also called as **external nodes** or **terminal nodes**.

8. Level:

- In a tree, **each step** from top to bottom is called as **level of a tree.**
- The level count starts with 0 and increments by 1 at each level or step





Basic Tree Terminologies

9. Height:

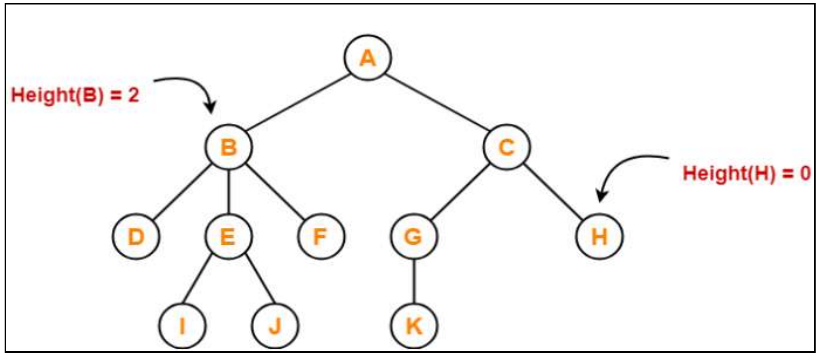
- **Total number of edges** that lies on the longest path from **any leaf node to a particular node is called as height of that node.**
- **Height of a tree** is the height of root node.
- Height of all leaf nodes = 0

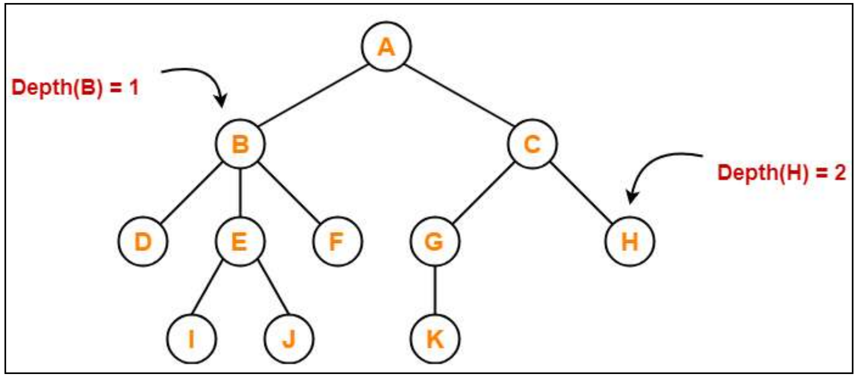
10.Depth:

- **Total number of edges from root node to a particular node** is called as **depth of that node**.
- **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
- Depth of the root node = 0

11.Subtree:

- In a tree, **each child from a node forms a subtree** recursively.
- Every child node forms a subtree on its parent node.





Representation of Binary Tree

Representation of Binary Tree There are two representations used to implement binary trees.

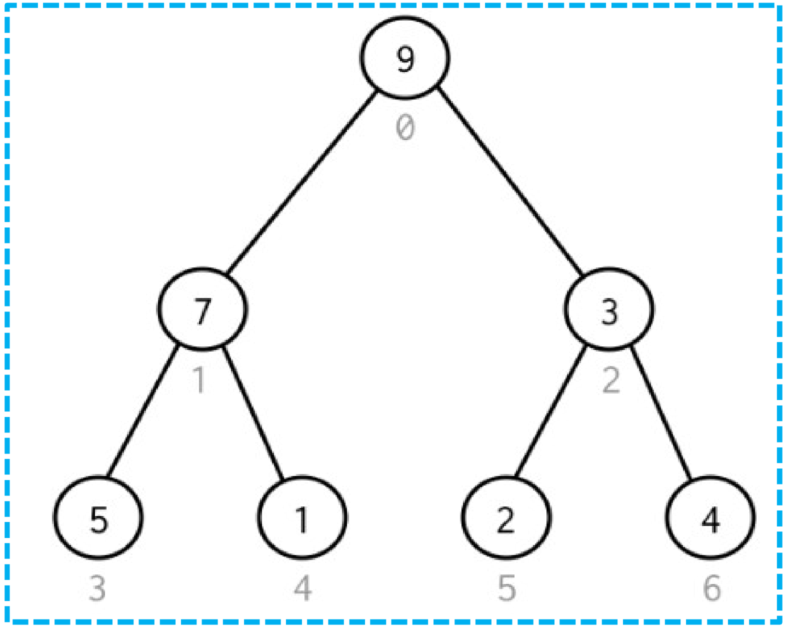
i) Array Representation

ii) Linked list Representation

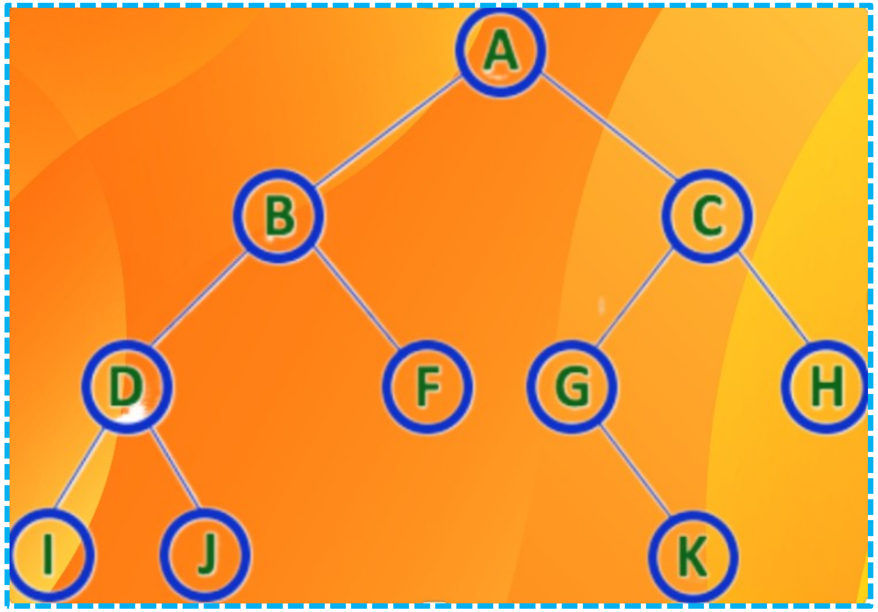
i)Array Representation:

- ✓ In array representation of a binary tree, **we use one-dimensional array (1-D Array) to represent a binary tree.**
- ✓ The main **drawback** of array representation is **wastage of memory** when there are many missing elements.

i)Array Representation:



9	7	3	5	1	2	4
0	1	2	3	4	5	6



A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

i)How to determine left child or Right child or Root node:

LeftChild(node):

- The **left child of node(n)** is at **position (2i+1)**
- Note: Here **'i' is the index position** of **node.**

RightChild(node):

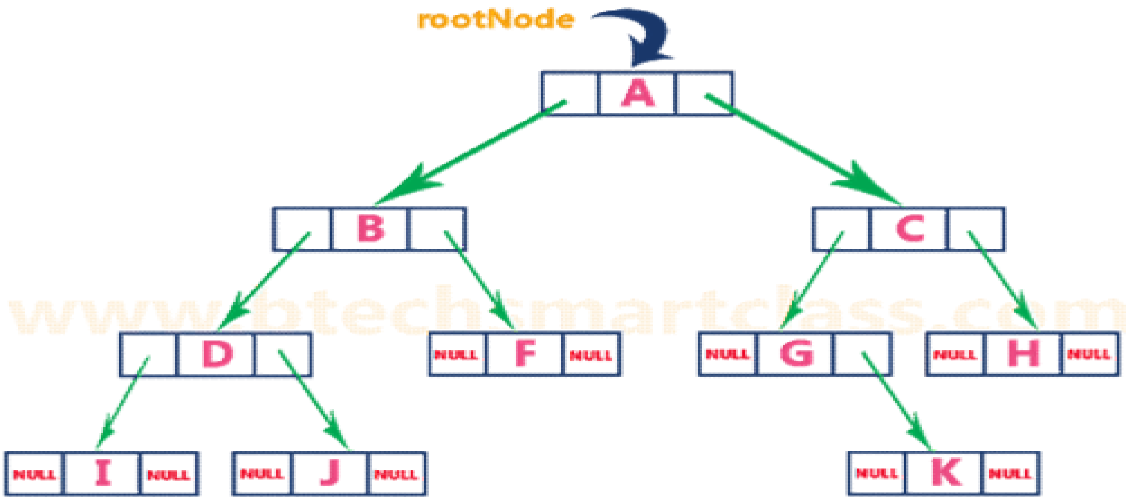
- The **right child of node(n)** is at **position (2i+2)**
- Note: Here **'i' is the index position** of **node.**

Root(node):

- The root of **Root(n)** is at position **(i-1)/2**
- Note: Here 'n' is the index position of node.

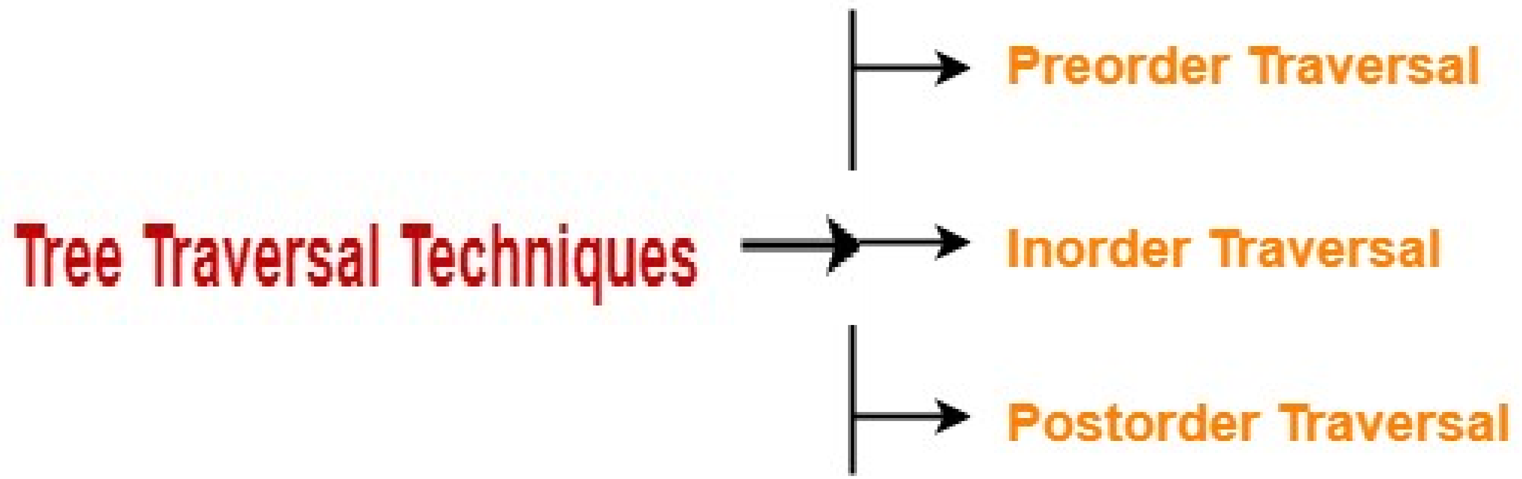
ii) Linked list Representation:

- We **use a double linked** list to **represent a binary tree.**
- In a double linked list, every node consists of three fields.
- **First** field for storing **left child address**, **second** for storing **actual data** and **third** for storing **right child address**.



Tree Traversal

- Tree Traversal refers to the process of **visiting each node in a tree data structure exactly once.**



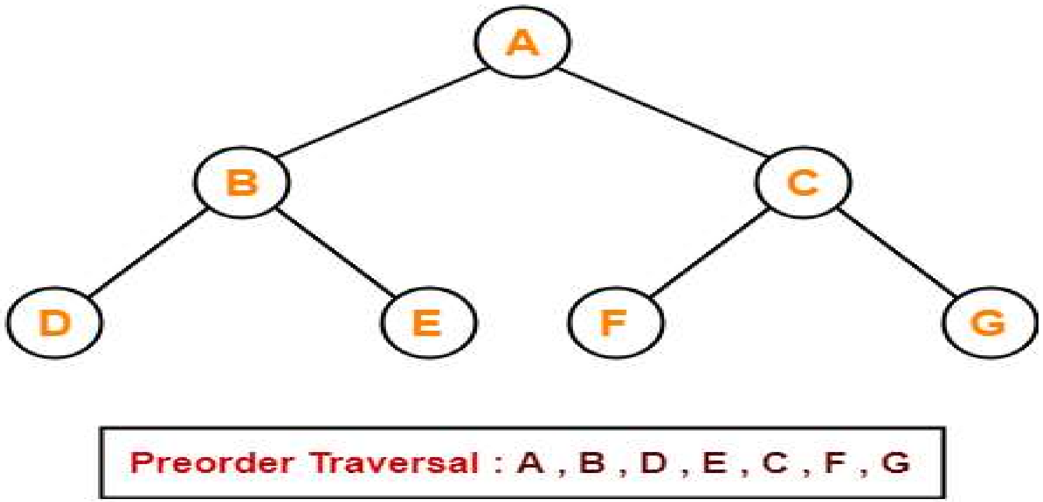
i. Preorder Traversal

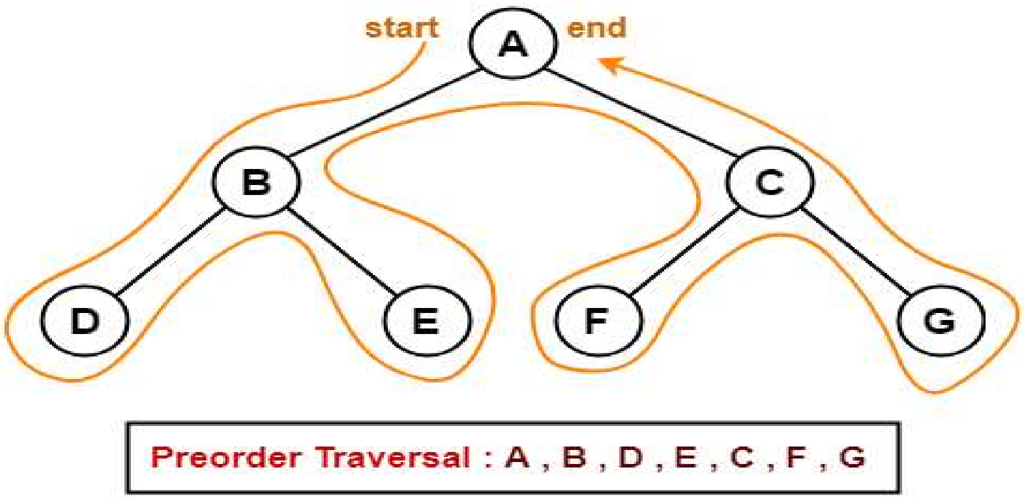
- It means that, first root node is visited after that the left subtree is traversed(Visited) recursively, and finally, right subtree is recursively traversed(Visited).

Algorithm:

- Visit the root node
- Traverse the left sub tree (or) Visit all the nodes in the left subtree
- Traverse the right sub tree (or) Visit all the nodes in the right subtree

Root → Left → Right





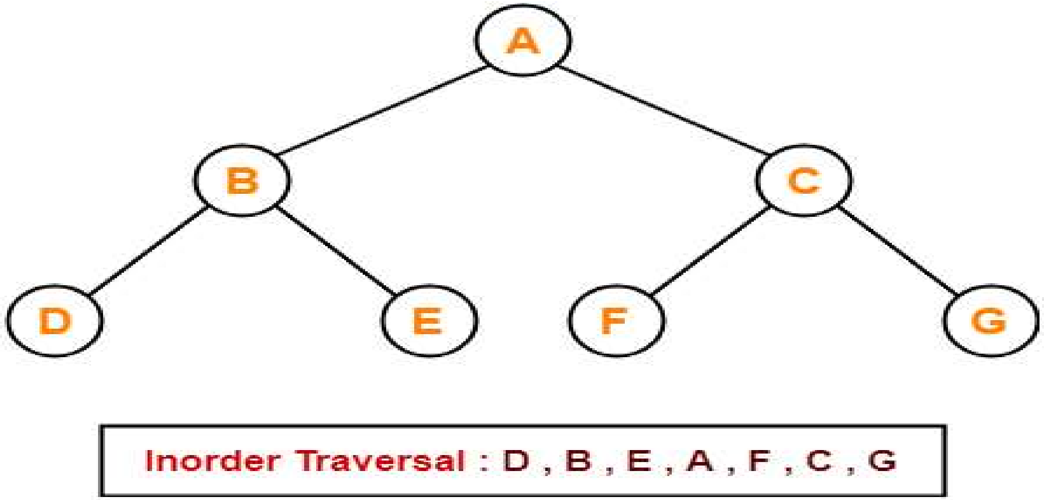
ii. Inorder Traversal

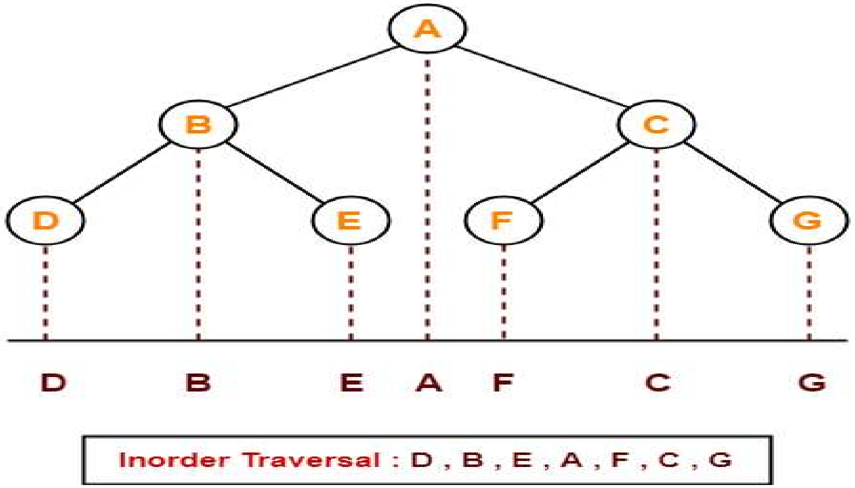
- In this traversal, the **left child node** is visited first, then the **root node** is visited and **later** we go for visiting the **right child node.**

Algorithm:

- **Traverse** the **left sub tree**
- **Visit** the **root**
- **Traverse** the **right sub tree**

Left → Root → Right





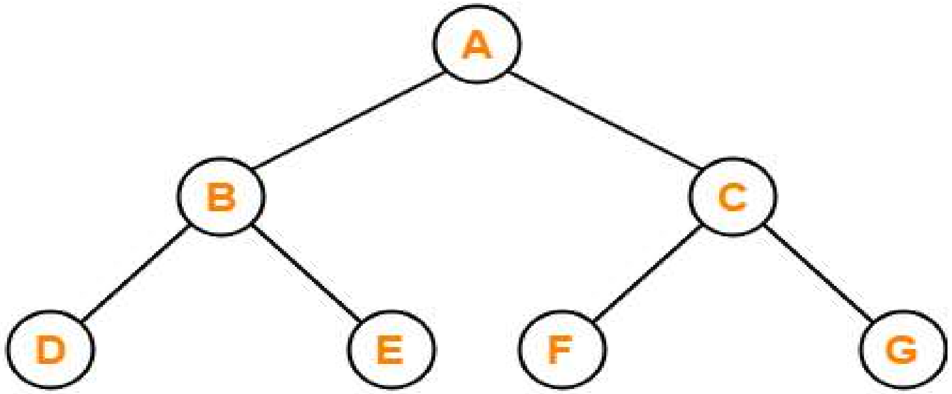
3. Postorder Traversal

- In this traversal, **left child node** is **visited first**, then its **right child** and **then** its **root node.** This is recursively performed until the right most node is visited.

Algorithm:

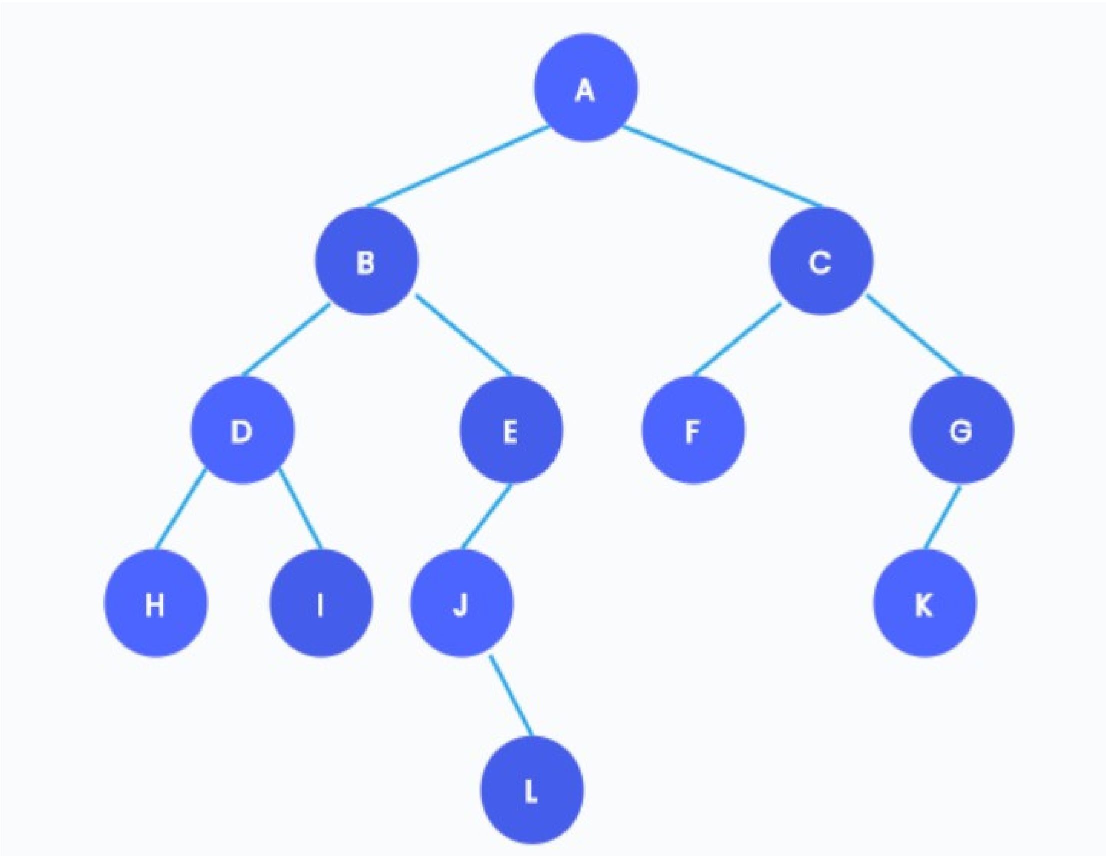
- **Traverse** the **left sub tree**
- **Traverse** the **right sub tree**
- **Visit** the **root**

Left → Right → Root



Postorder Traversal : D , E , B , F , G , C , A

Example 1 for Tree Traversal

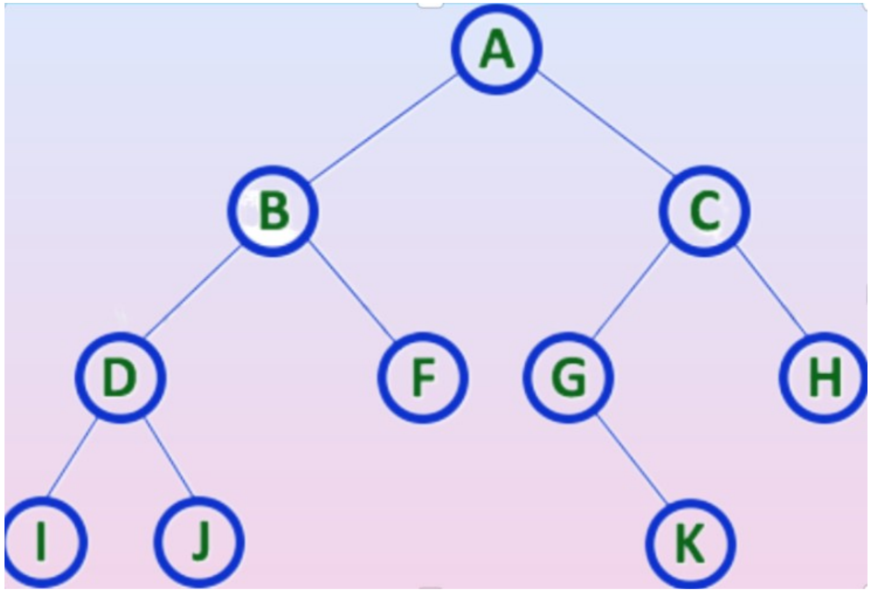


In-order Traversal - H, D, I, B, J, L, E, A, F, C, K ,G

Pre-order Traversal - A, B, D, H, I, E, J, L, C, F, G, K

Post-order Traversal - H, I, D, L, J, E, B, F, K, G, C, A

Example -2 for Tree Traversal



In-Order Traversal for above example of binary tree is

I - D - J - B - F - A - G - K - C - H

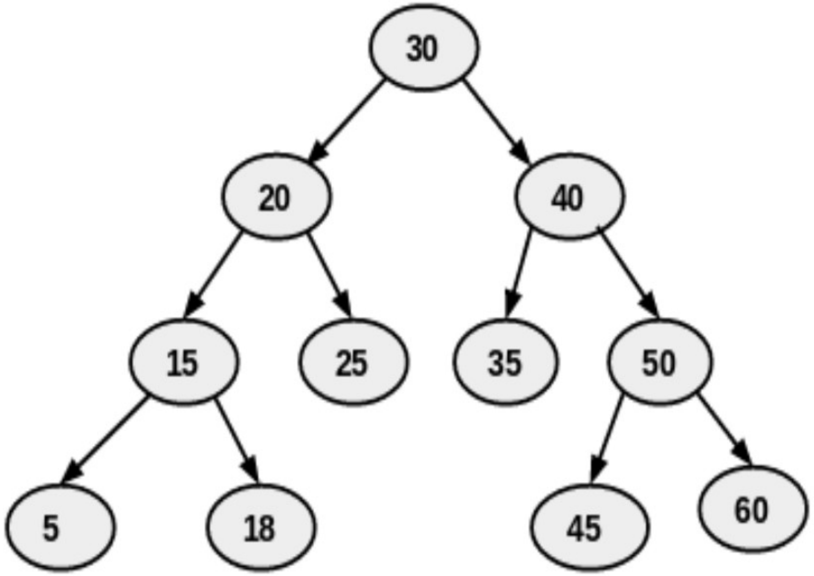
Pre-Order Traversal for above example binary tree is

A - B - D - I - J - F - C - G - K - H

Post-Order Traversal for above example binary tree is

I - J - D - F - B - K - G - H - C - A

Example -3 for Tree Traversal



{5 , 15 , 18 , 20 , 25 , 30 , 35 , 40 , 45 , 50 , 60}

{30 , 20 , 15 , 5 , 18 , 25 , 40 , 35 , 50 , 45 , 60}

{5 , 18 , 15 , 25 , 20 , 35 , 45 , 60 , 50 , 40 , 30}

Construction of Binary Tree from Tree Traversals

- Construction of a binary tree when the tree traversals is given:

    i. **Tree can be constructed when INORDER and PREORDER traversals is given.**

    ii. **Tree can be constructed when INORDER and POSTORDER traversals is given.**

    iii. **Tree can be constructed when PREORDER and POSTORDER traversals is given.**

How to Construct Tree From INORDER and PREORDER Traversals

Step-1: Make the first element in the PREORDER traversal as ROOT node.

Step-2: Read next element from PREORDER.

Step-3: If the element appear left of root in INORDER the add that element to the left of ROOT node otherwise add the element to the right of ROOT node.

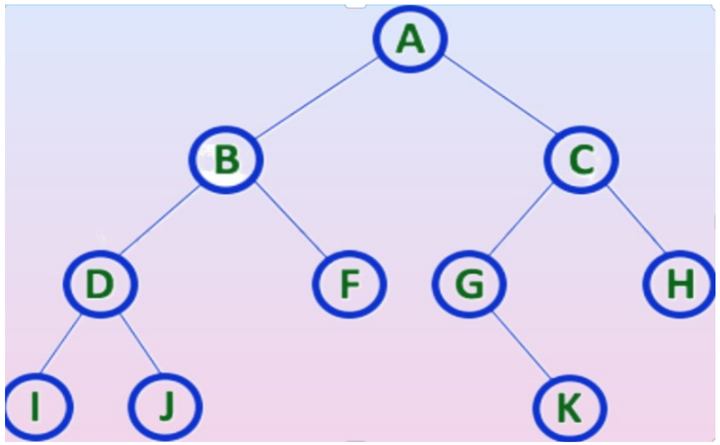
Step-4: Recursively continue the same process until completion of all the elements in PREORDER.

Pre-Order Traversal for above example binary tree is

A - B - D - I - J - F - C - G - K - H

In-Order Traversal for above example of binary tree is

I - D - J - B - F - A - G - K - C - H



How to Construct Tree From INORDER and POSTORDER Traversals

Here, From the POSTORDER we will get ROOT node information, from the INORDER we will get left and right subtree of the ROOT.

Step-1: Make the last element in the POSTORDER traversal as ROOT node.

Step-2: Read predecessors(Previous) element from POSTORDER.

Step-3: If the element appear left of root in INORDER the add that element to the left of root otherwise add the element to the right of root.

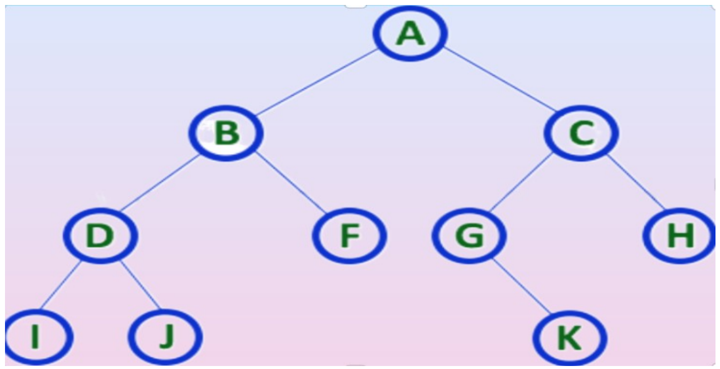
Step-4: Recursively continue the same process until completion of all the elements in POSTORDER

Post-Order Traversal for above example binary tree is

I - J - D - F - B - K - G - H - C - A

In-Order Traversal for above example of binary tree is

I - D - J - B - F - A - G - K - C - H



How to Construct Tree From PREORDER and POSTORDER Traversals

Step-1: Here, From the **PREORDER** and **POSTORDER**, we will **finalize the ROOT node.**

Step-2: Assume the **N1 as right child of Root** node and **N2 as left child of Root** node.

N1=**Predecessor** of root node in **POSTORDER**

N2=**Successor** of root node in **PREORDER**

Here, We **have 2-cases**:

Case-1:If **N1==N2** then the **tree is not Unique.**

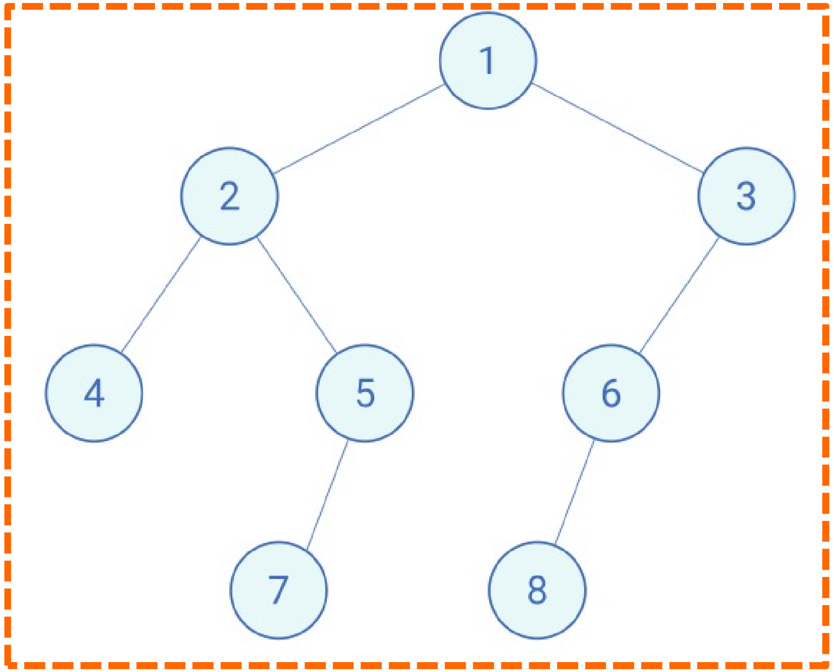
Case-2:If **N1!=N2** then **N1→Right Child** and **N2→ Left Child**

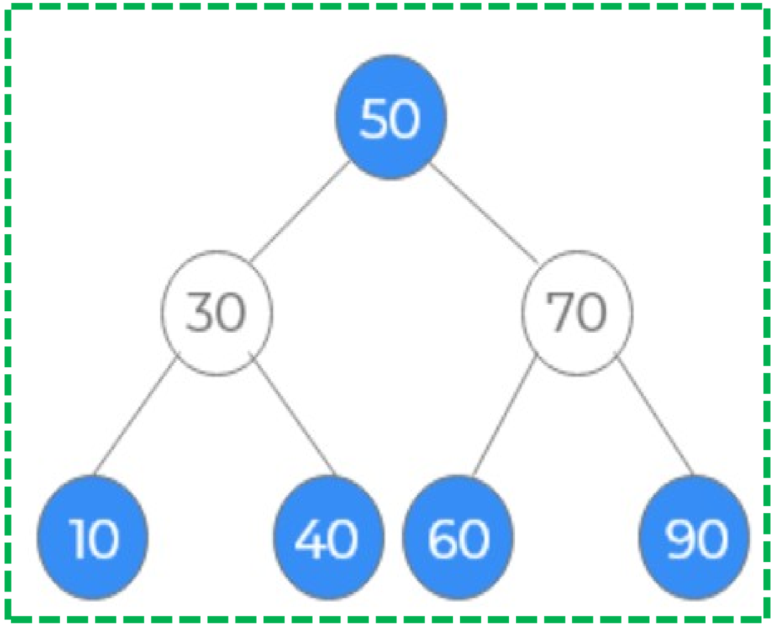
Step-3: We **have to repeat** the **same steps** to **construct the next levels** of the **tree.**

Examples

Preorder: [1, 2, 4, 5, 7, 3, 6, 8]
Inorder: [4, 2, 7, 5, 1, 8, 6, 3]

Inorder - [10 30 40 50 60 70 90]
Postorder - [10 40 30 60 90 70 50]





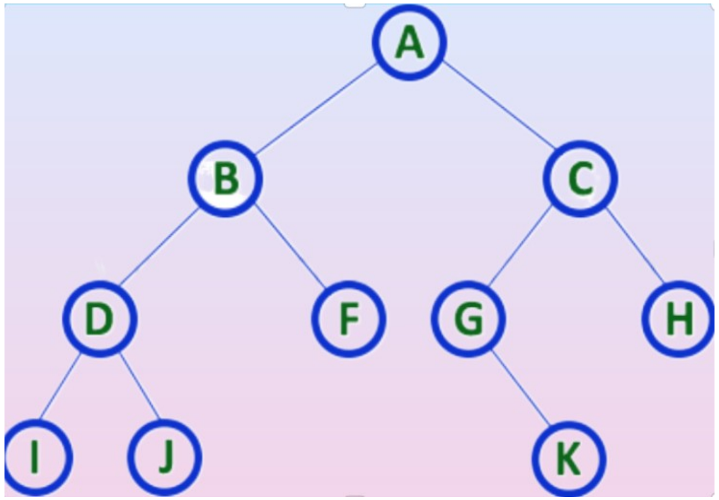
How to Construct Tree From PREORDER and POSTORDER Traversals

Pre-Order Traversal for above example binary tree is

A - B - D - I - J - F - C - G - K - H

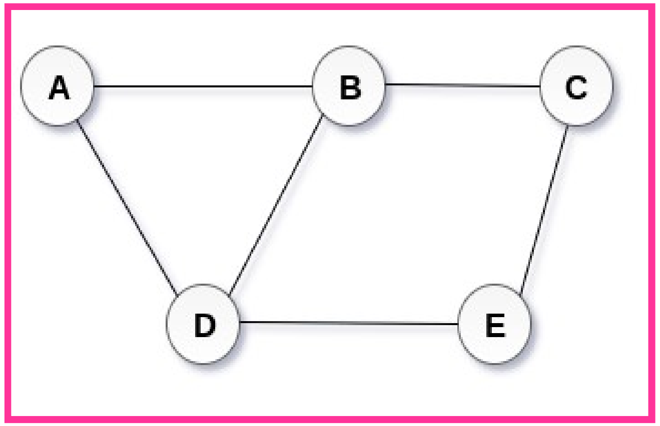
Post-Order Traversal for above example binary tree is

I - J - D - F - B - K - G - H - C - A

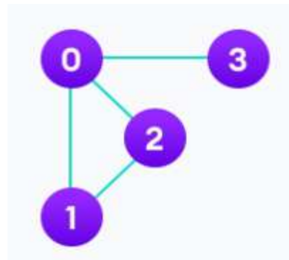


Graphs

- A **graph G** is simply a **set V of vertices** and a **collection edges(E).**
- It is a collection **of *nodes*** *and* ***arcs.***
- A **graph G** can be **defined** as an ordered set **G(V, E)** where **V represents** the set of **vertices** and **E represents** the set of **edges.**
- A Graph **G(V, E)** with **5 vertices (A, B, C, D, E)** and **six edges ((A,B), (B,C), (C,E), (E,D), (D,B), (D,A))** is shown in the following figure.



V = {0, 1, 2, 3}
E = {(0,1), (0,2), (0,3), (1,2)}
G = {V, E}



Graphs

- **Edges** in a **graph** are either **directed or undirected.**
- An **edge (u,v)** is said to be **directed** from u to v if the pair (u,v) **is ordered**, with **u preceding v.**
- An **edge (u,v)** is said to be **undirected** if the pair (u,v) is **not ordered.**
- **Undirected edges** are **sometimes denoted** with **set notation, as {u,v},** but for simplicity we use the pair notation (u,v), noting that in the undirected case **(u,v) is the same as (v,u).**

Types of Graphs:

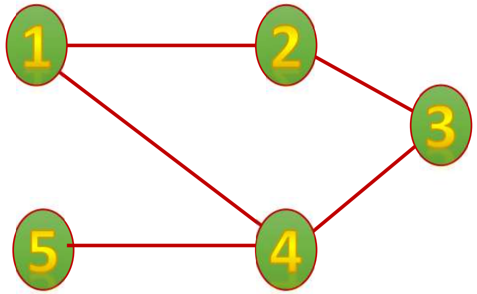
i. **Undirected graph**

ii. **Directed graph**

iii. **Mixed graph**

Types of Graphs

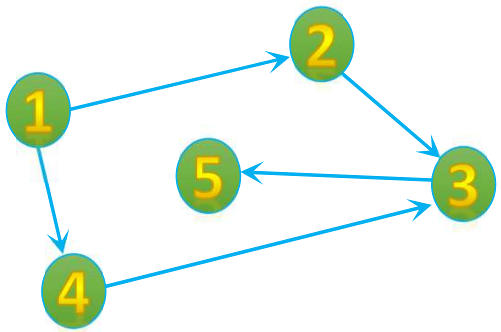
Undirected graph:



- If **all the edges** in a **graph are undirected**, then we say the graph is an **undirected graph.**

Directed graph:

- a **directed graph, also called a digraph, is a graph whose edges are all directed.**



Mixed graph:

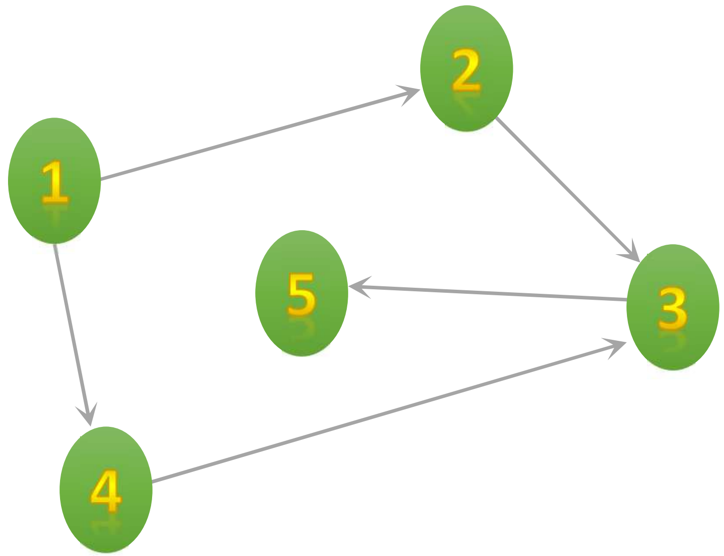
- A **graph** that has **both directed** and **undirected edges** is often called a **mixed graph.**

Types of Graphs

- The **two vertices joined** by an **edge** are called the **end vertices (endpoints)** of the edge.
- If **an edge is directed**, its first **endpoint is its origin** and the other is the **destination** of the edge.
- **Two vertices u and v** are said to be **adjacent** if there is an **edge whose end vertices are u and v.**
- An **edge is said** to be **incident** on **a vertex**.
- The **outgoing edges** of a vertex are the **directed edges** whose **origin** is that **vertex.**
- The **incoming edges** of a vertex are the **directed edges whose destination** is that **vertex.**

Types of Graphs

- The **degree of a vertex v**, denoted **deg(v),** is the **number of incident edges of v.**
- The **in-degree** and **out-degree of a vertex v** are the **number of** the **incoming** and **outgoing edges of vertex.**
- These are **denoted indeg(v)** and **outdeg(v)**.



Degree of 3 is 3

In Degree of 3 is 2

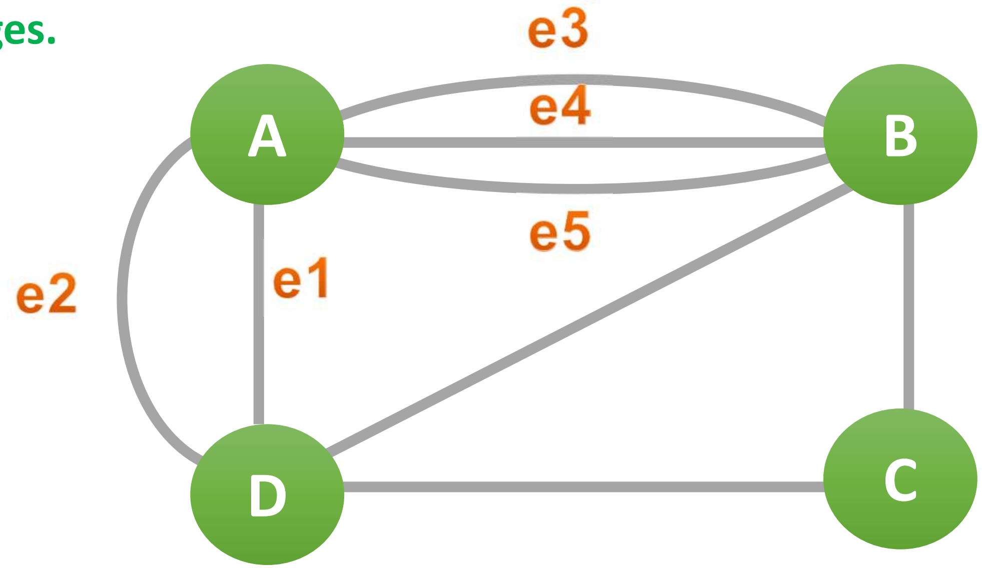
Out Degree of 3 is 1

Degree of 4 is 2

In Degree of 4 is 1

Types of Graphs

- Allowing for two or more undirected edges or directed edges between the same end vertices. Such edges are called parallel edges.

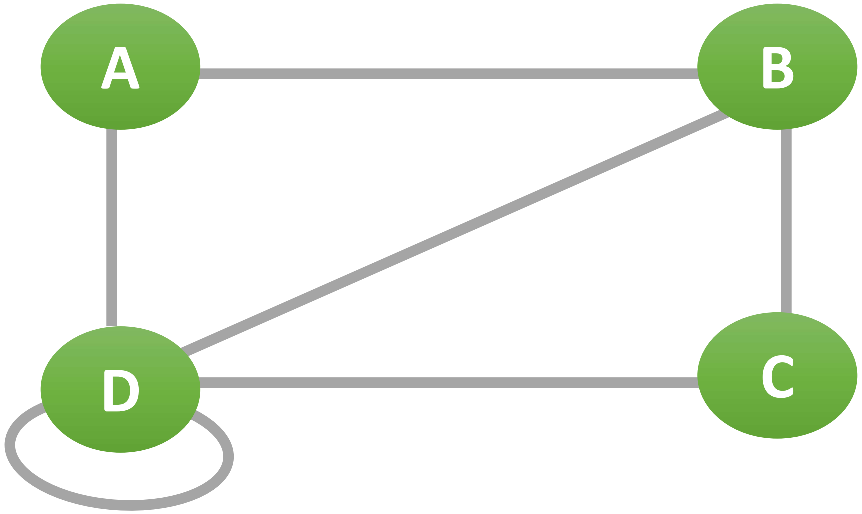


- Graph Representation techniques

  1. Adjacency Matrix
  2. Incidence Matrix
  3. Adjacency List

Representation of Graph

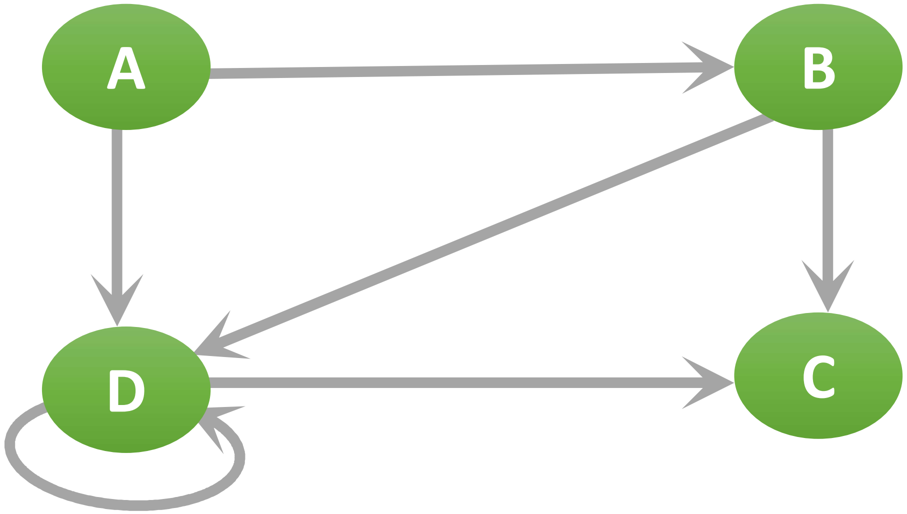
1. Adjacency Matrix



$$
\begin{array}{c c}
 & \begin{array}{cccc} A & B & C & D \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}
\end{array}
$$

Representation of Graph

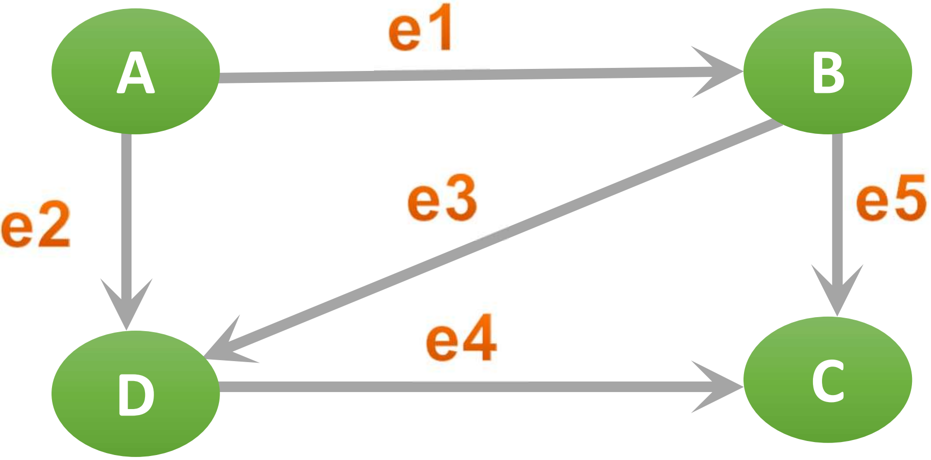
1. Adjacency Matrix



$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

Representation of Graph

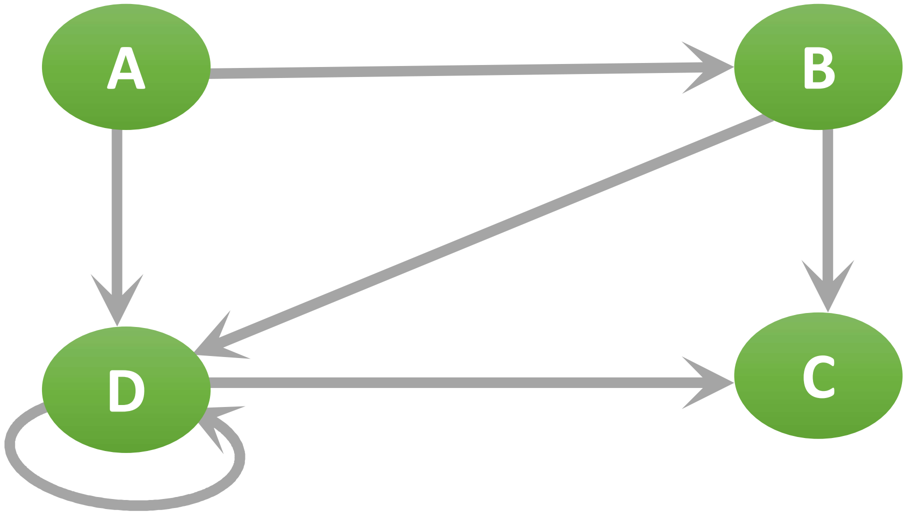
2. Incidence Matrix



$$
\begin{array}{c c}
 & \begin{array}{ccccc} e1 & e2 & e3 & e4 & e5 \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \end{array} &
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & -1 & -1 \\
0 & -1 & -1 & 1 & 0
\end{bmatrix}
\end{array}
$$

Representation of Graph

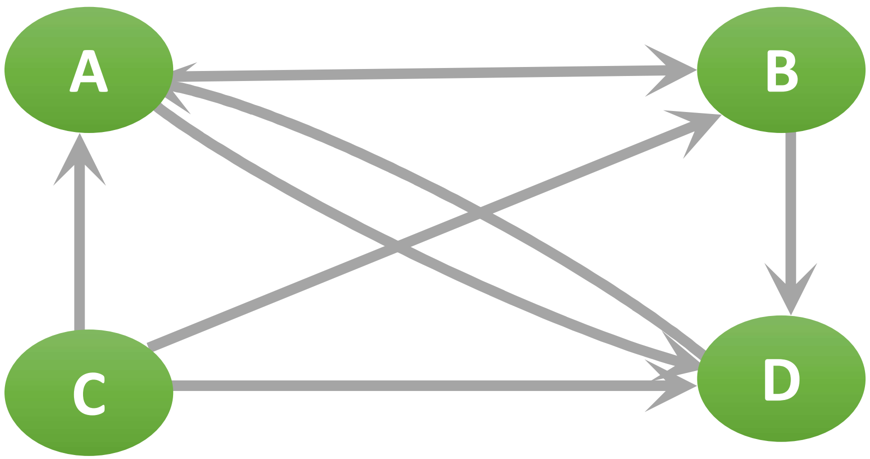
3. Adjacency List



B	D
C	D
C	D

Representation of Graph

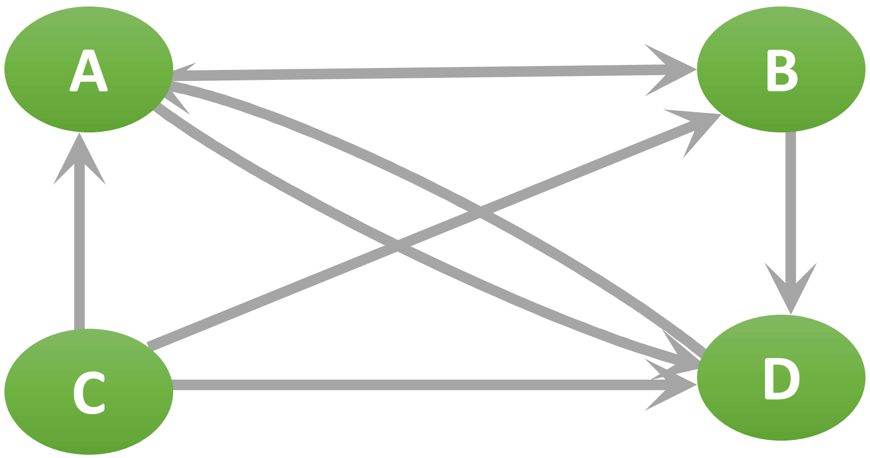
3. Adjacency List

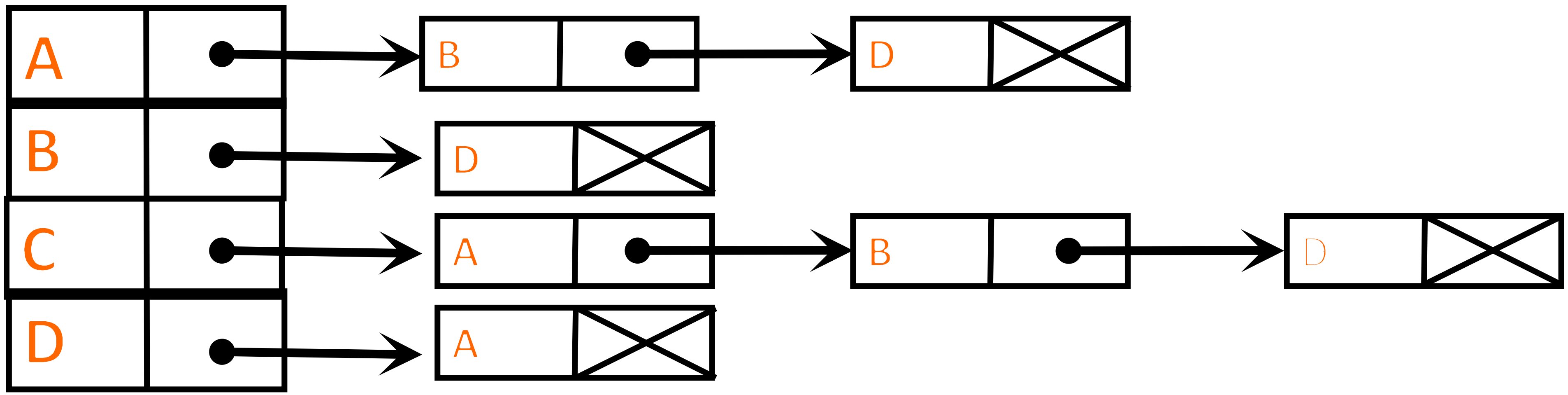


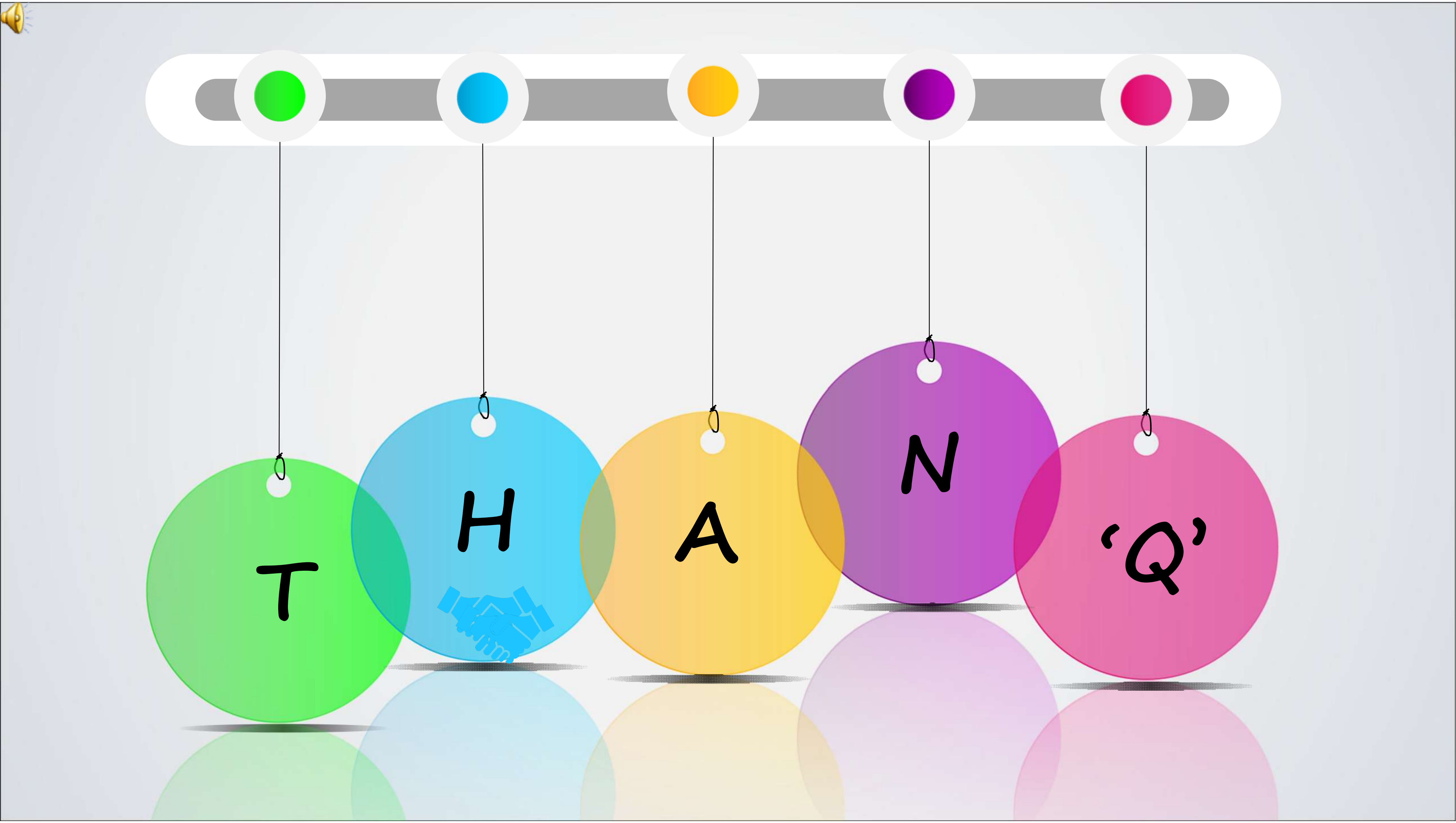
B	D	
D		
A	B	D
A		

Representation of Graph

3. Adjacency List






T
H
A
N
'Q'