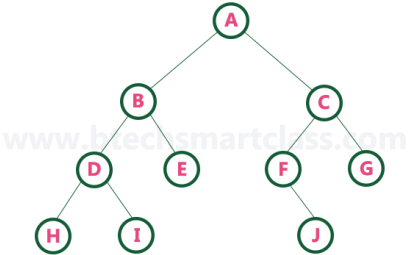


St. Peter's Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. <b>QUESTION BANK</b>				Dept.	:	CSE, CSM, CSD, CSC
				Academic Year 2023-24		
Subject Code	:	AS22-05ES07	Subject	:	Data Structures	
Class/Section	:	B.Tech.	Year	:	I	Semester : II

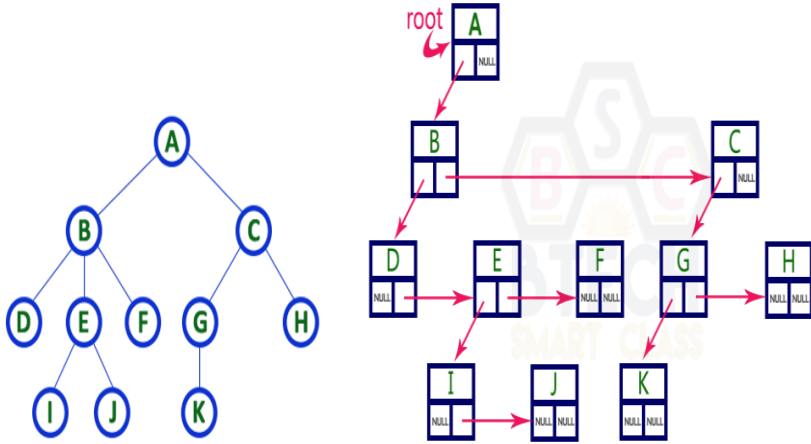
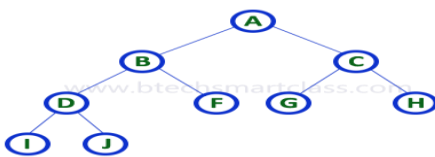
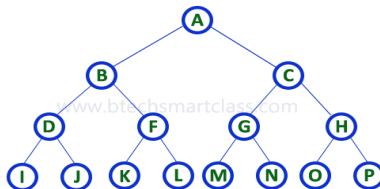
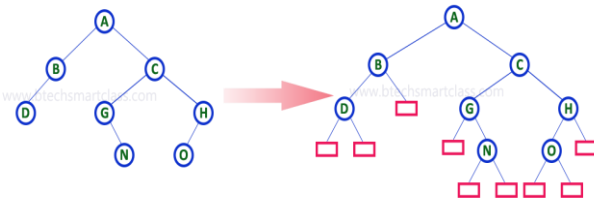
BLOOMS LEVEL					
Remember	L1	Understand	L2	Apply	L3
Analyze	L4	Evaluate	L5	Create	L6

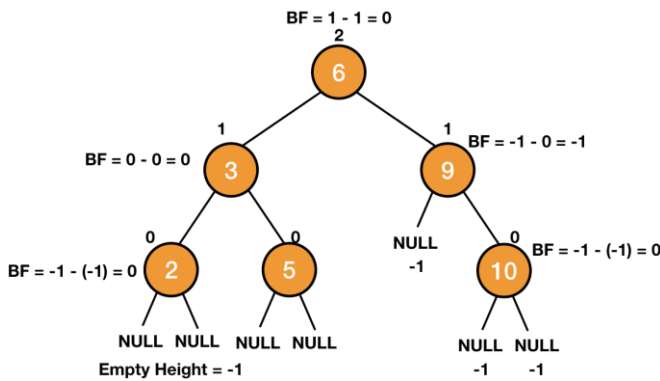
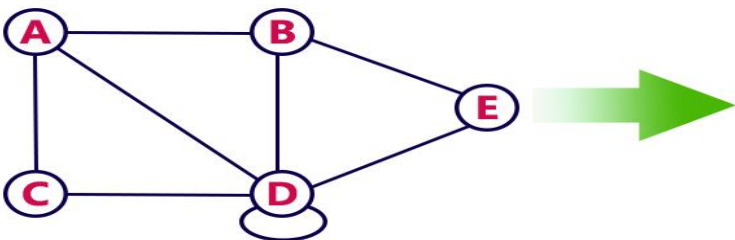
\*\*\*\*\*

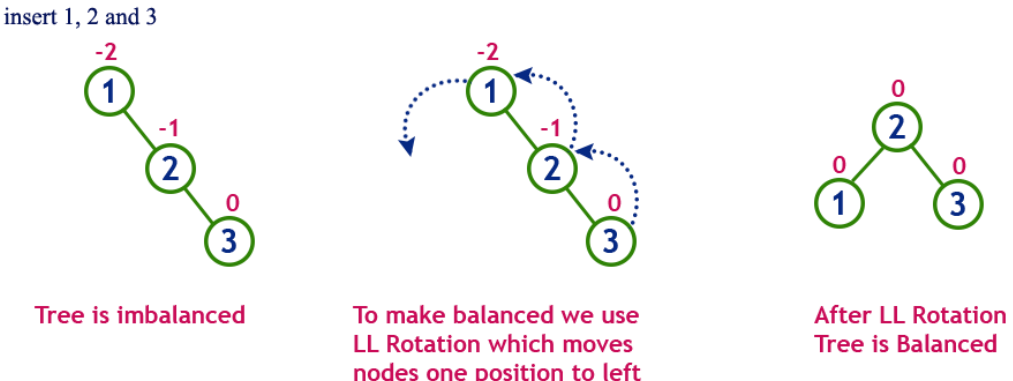
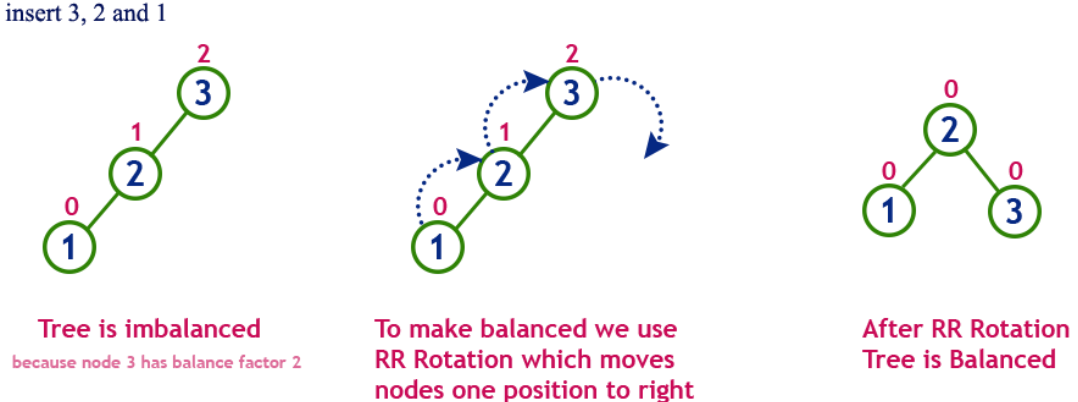
Q. No	Question (s)	Marks	BL	CO
UNIT - I				
1	a) What is Binary Tree? Draw the diagram.?	1M	L1	C123.5
	<p>In a normal tree, every node can have any number of children. A binary tree is a special type of tree data structure in which every node can have a <b>maximum of 2 children</b>. One is known as a left child and the other is known as right child. In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.</p> 			
	b) List the various operations in binary search tree.	1M	L2	C123.5
	<p>Binary search tree is a binary tree in which all the nodes in the left subtree of any node contains smaller values and all the nodes in the right subtree of any node contains larger values. <b>Every binary search tree is a binary tree but every binary tree need not to be binary search tree.</b></p> <ol style="list-style-type: none"> <li>1.Search</li> <li>2.Insertion</li> <li>3.Deletion</li> </ol>			
	c) What are the tree traversal methods.	1M	L1	C123.5

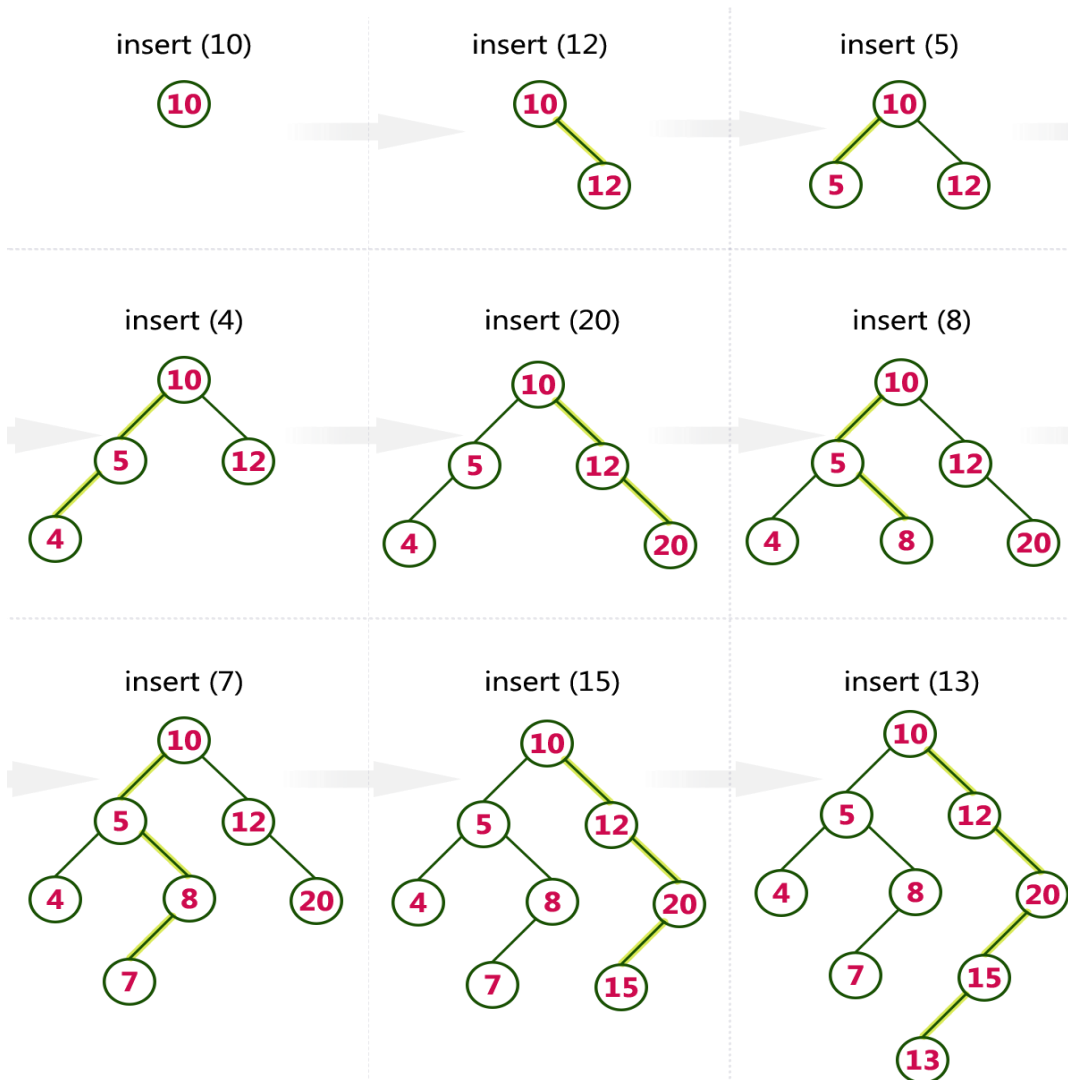
	1. In - Order Traversal 2. Pre - Order Traversal 3. Post - Order Traversal			
	<b>d) Write the balance factor formula for AVL tree</b>	<b>1M</b>	<b>L1</b>	<b>C123.5</b>
	Balance factor = height of Left Subtree – height of Right Subtree OR Balance factor = height of Right Subtree – height of Right Subtree			
	<b>e) Define Graph traversal.</b>	<b>1M</b>	<b>L1</b>	<b>C123.5</b>
	Graph traversal is a technique used for a searching vertex in a graph. There are two graph traversal techniques and they are as follows: 1.BFS stands for Breadth First Search 2.DFS stands for Depth First Search.			

2	<b>a) Differentiate between BFS and DFS</b>		<b>3M</b>	<b>L2</b>	<b>C123.5</b>
	Parameters	BFS	DFS		
	Stands for	BFS stands for Breadth First Search.	DFS stands for Depth First Search.		
	Data Structure	BFS (Breadth First Search) uses Queue data structure for finding the shortest path.	DFS (Depth First Search) uses Stack data structure.		
	Definition	BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.		
	Conceptual Difference	BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.		
	Approach used	It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).		
	Suitable for	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.		

Applications	bipartite graphs, shortest paths, etc.	acyclic graphs		
<p><b>b) Show the Left Child - Right Sibling Representation of a given Tree.</b></p> <div>  </div>		3M	L2	C123.5
<p><b>c) List the varoius types of a tree.</b></p>		3M	L1	C123.5
<p>1. Strictly Binary Tree: A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree.</p> <div>  </div> <p>2. Complete Binary Tree: A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.</p> <div>  </div> <p>3. Extended Binary Tree: The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.</p> <div>  </div>				

d) Write a short note on AVL tree.	3M	L1	C123.5																																				
<p>AVL tree is a height-balanced binary search tree. That means, an AVL tree is also a binary search tree but it is a balanced tree. A binary tree is said to be balanced if, the difference between the heights of left and right subtrees of every node in the tree is either -1, 0 or +1. In other words, a binary tree is said to be balanced if the height of left and right children of every node differ by either -1, 0 or +1.</p> <p><b>Balance factor = height of Left Subtree – height of Right Subtree</b></p> <div></div>																																							
e) Enumerate the adjacent matrix representation of graph data structure.	3M	L3	C123.5																																				
<p>In this representation, the graph is represented using a matrix of size total number of vertices by a total number of vertices. That means a graph with 4 vertices is represented using a matrix of size 4X4. In this matrix, both rows and columns represent vertices. This matrix is filled with either 1 or 0. Here, 1 represents that there is an edge from row vertex to column vertex and 0 represents that there is no edge. For example, consider the following undirected graph representation</p> <div></div> <div><table><tr><td></td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td></tr><tr><td>A</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>B</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>C</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>D</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>E</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table></div>					A	B	C	D	E	A	0	1	1	1	0	B	1	0	0	1	1	C	1	0	0	1	0	D	1	1	1	1	1	E	0	1	0	1	0
	A	B	C	D	E																																		
A	0	1	1	1	0																																		
B	1	0	0	1	1																																		
C	1	0	0	1	0																																		
D	1	1	1	1	1																																		
E	0	1	0	1	0																																		

3	a) Explain AVL tree LL & RR rotations	5M	L2	C123.5
	<p>In AVL tree, after performing operations like insertion and deletion we need to check the <b>balance factor</b> of every node in the tree. Whenever the tree becomes imbalanced due to any operation we use <b>rotation</b> operations to make the tree balanced.</p> <p>In <b>LL Rotation</b>, every node moves one position to left from the current position. To understand LL Rotation, let us consider the following insertion operation in AVL Tree.</p> <p>insert 1, 2 and 3</p>  <p>Tree is imbalanced</p> <p>To make balanced we use LL Rotation which moves nodes one position to left</p> <p>After LL Rotation Tree is Balanced</p> <p>In <b>RR Rotation</b>, every node moves one position to right from the current position. To understand RR Rotation, let us consider the following insertion operation in AVL Tree.</p> <p>insert 3, 2 and 1</p>  <p>Tree is imbalanced because node 3 has balance factor 2</p> <p>To make balanced we use RR Rotation which moves nodes one position to right</p> <p>After RR Rotation Tree is Balanced</p>			
	b) Construct the Binary Search Tree by inserting the following sequence of numbers 10,12,5,4,20,8,7,15 and 13	5M	L5	C123.5

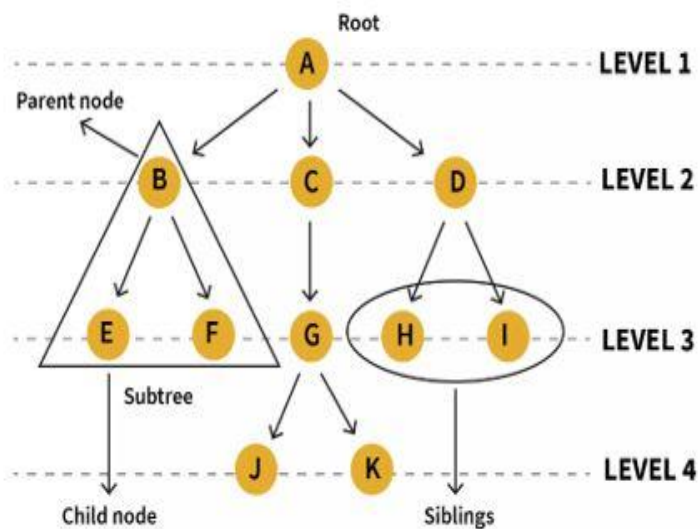


c) Describe the terms used in tree data structure.

5M

L1

C123.5



Terminology	Description	Diagram
Root	Root node is a node from which the entire tree originates. It does not have a parent	Node A
Parent Node	An immediate predecessor of any node is its parent node.	B is parent of E & F
Child Node	All immediate successors of a node are its children. The relationship between the parent and child is considered as the parent-child relationship.	F & E are children of B
Leaf	Node which does not have any child is a leaf. Usually the boundary nodes of a tree or last nodes of the tree are the leaf or collectively called leaves of the tree.	E, F, J, K, H, I are the leaf nodes.
Siblings	Siblings in real life means people with the same parents, similarly in the case of trees, nodes with common parents are considered to be siblings.	H&I are siblings
Height of Node	Height of a node represents the number of edges on the longest path between that node and a leaf.	A, C, G, K form a height. Height of A is no. of edges between A and K, which is 3. Similarly the height of G is 1 as it has just one edge until the next leaf node.
Levels of node	Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on	Level of H, I & J is 3. Level of D, E, F & G is 2
Degree of Node	Degree of a node implies the number of child nodes a node has.	Degree of D is 2 and of C is 3
Internal Node	A node that has at least one child is known as an internal node.	All the nodes except E, F, J, K, H, I are internal.
Ancestor node	An ancestor or ancestors to a node are all the predecessor nodes from root until that node. I.e. any parent or grandparent and so on of a specific node are its ancestors.	A, C & G are ancestor to K and J nodes
Descendant	Immediate successor of a node is its descendant.	K is descendent of G
Sub tree	Descendants of a node represent subtree. Tree being a recursive data structure can contain many subtrees inside of it.	Nodes B, E, F represent one subtree.

**d) Analyze the steps involved in the insertion operation of Binary Search Tree.**

5M

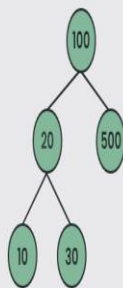
L4

C123.5

In a binary search tree, the insertion operation is performed with  $O(\log n)$  time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

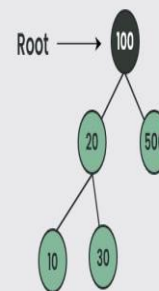
- **Step 1** - Create a newNode with given value and set its **left** and **right** to **NULL**.
- **Step 2** - Check whether tree is Empty.
- **Step 3** - If the tree is **Empty**, then set **root** to **newNode**.
- **Step 4** - If the tree is **Not Empty**, then check whether the value of newNode is **smaller** or **larger** than the node (here it is root node).
- **Step 5** - If newNode is **smaller** than or **equal** to the node then move to its **left** child. If newNode is **larger** than the node then move to its **right** child.
- **Step 6** - Repeat the above steps until we reach to the **leaf** node (i.e., reaches to NULL).
- **Step 7** - After reaching the leaf node, insert the newNode as **left child** if the newNode is **smaller or equal** to that leaf node or else insert it as **right child**.

Consider The Following BST



$X = 40$  ( The Node To Be Inserted )

STEP 1: Comparing X with Root Node



Since 100 Is Greater Than 40.  
Move Pointer To The Left Child ( 20 )

Insertion In BST

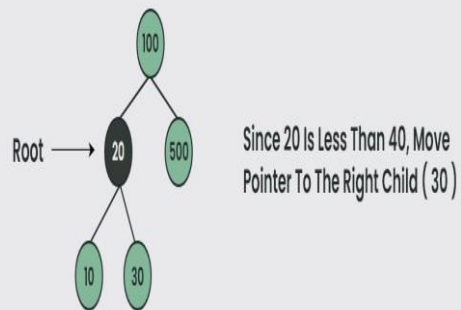


Insertion In BST





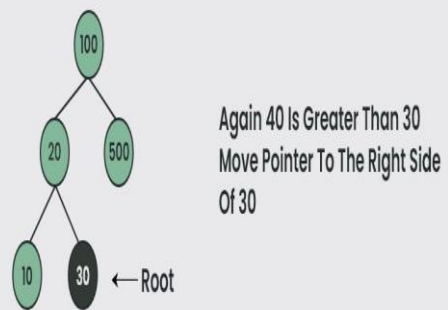
### STEP 2 : Comparing X with left child of root node



Insertion In BST



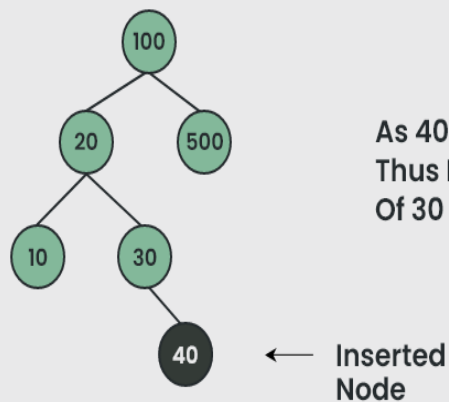
### STEP 3 : Comparing x with the right child of 20



Insertion In BST



### STEP 4 : Insert item to the right of 30



Insertion In BST



e) Explain the concept of incidence matrix and adjacency list of graph representations data structure.

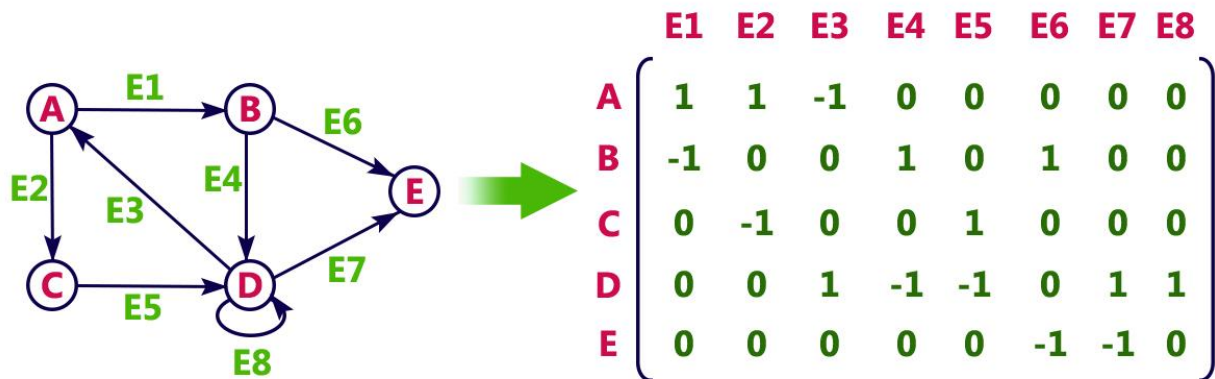
5M

L2

C123.5

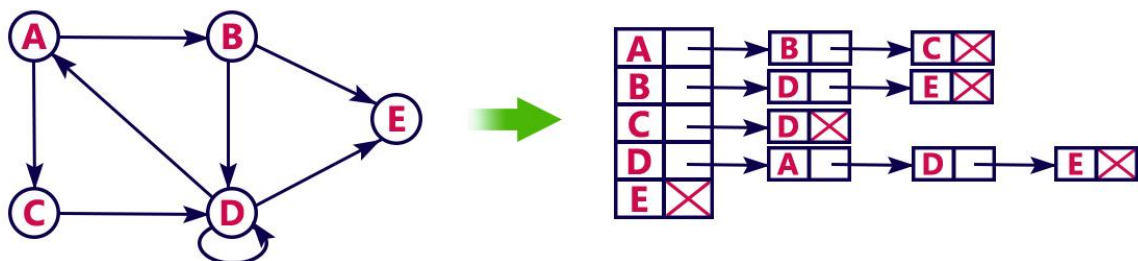
### Incident Matrix

In this representation, the graph is represented using a matrix of size total number of vertices by a total number of edges. That means graph with 4 vertices and 6 edges is represented using a matrix of size 4X6. In this matrix, rows represent vertices and columns represents edges. This matrix is filled with 0 or 1 or -1. Here, 0 represents that the row edge is not connected to column vertex, 1 represents that the row edge is connected as the outgoing edge to column vertex and -1 represents that the row edge is connected as the incoming edge to column vertex. For example, consider the following directed graph representation



### Adjacency List

In this representation, every vertex of a graph contains list of its adjacent vertices. For example, consider the following directed graph representation implemented using linked list



4	a) Explain the operations on AVL Tree.	10M	L2	C123.1
<p>The following operations are performed on AVL tree...</p> <ol style="list-style-type: none"> <li>1. Search</li> <li>2. Insertion</li> <li>3. Deletion</li> </ol> <p style="text-align: center;"><b>1. Search Operation in AVL Tree</b></p> <p>In an AVL tree, the search operation is performed with <b><math>O(\log n)</math></b> time complexity. The search operation in the AVL tree is similar to the search operation in a Binary search tree. We use the following steps to search an element in AVL tree...</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Read the search element from the user.</li> <li>• <b>Step 2</b> - Compare the search element with the value of root node in the tree.</li> <li>• <b>Step 3</b> - If both are matched, then display "Given node is found!!!" and terminate the function</li> <li>• <b>Step 4</b> - If both are not matched, then check whether search element is smaller or larger than that node value.</li> <li>• <b>Step 5</b> - If search element is smaller, then continue the search process in left subtree.</li> <li>• <b>Step 6</b> - If search element is larger, then continue the search process in right subtree.</li> <li>• <b>Step 7</b> - Repeat the same until we find the exact element or until the search element is compared with the leaf node.</li> <li>• <b>Step 8</b> - If we reach to the node having the value equal to the search value, then display "Element is found" and terminate the function.</li> <li>• <b>Step 9</b> - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.</li> </ul> <p style="text-align: center;"><b>2. Insertion Operation in AVL Tree</b></p> <p>In an AVL tree, the insertion operation is performed with <b><math>O(\log n)</math></b> time complexity. In AVL Tree, a new node is always inserted as a leaf node. The insertion operation is performed as follows...</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Insert the new element into the tree using Binary Search Tree insertion logic.</li> <li>• <b>Step 2</b> - After insertion, check the <b>Balance Factor</b> of every node.</li> <li>• <b>Step 3</b> - If the <b>Balance Factor</b> of every node is <b>0 or 1 or -1</b> then go for next operation.</li> </ul>				

- **Step 4** - If the **Balance Factor** of any node is other than **0 or 1 or -1** then that tree is said to be imbalanced. In this case, perform suitable **Rotation** to make it balanced and go for next operation.

**Example: Construct an AVL Tree by inserting numbers from 1 to 8.**

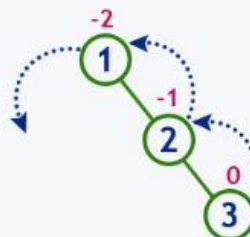
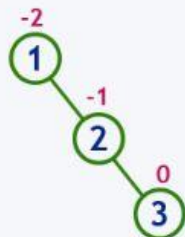
insert 1



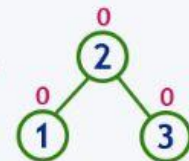
insert 2



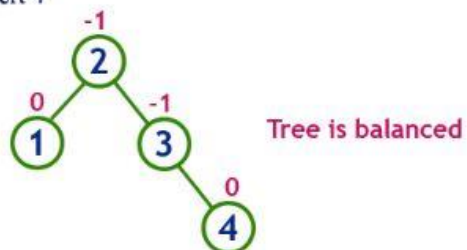
insert 3



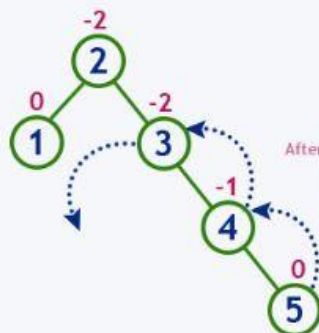
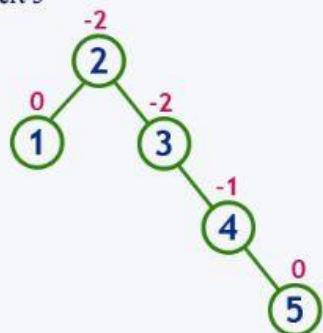
After LL Rotation



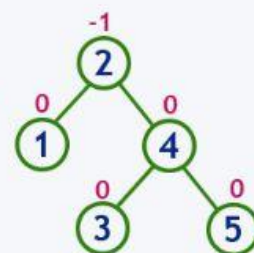
insert 4



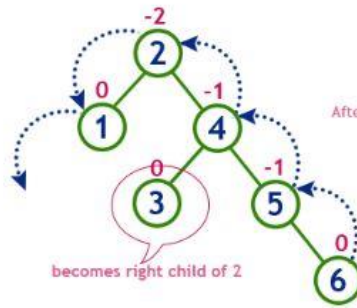
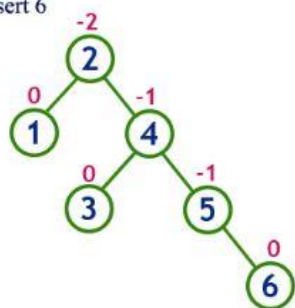
insert 5



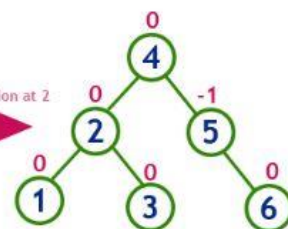
After LL Rotation at 3



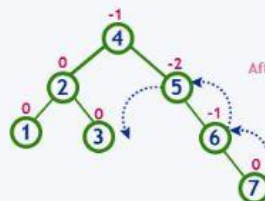
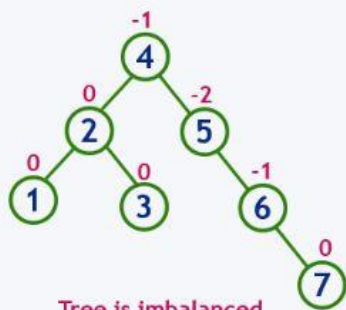
insert 6



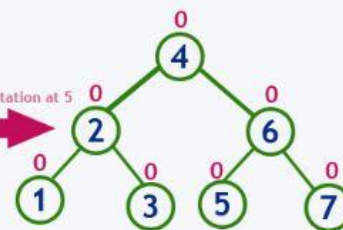
After LL Rotation at 2



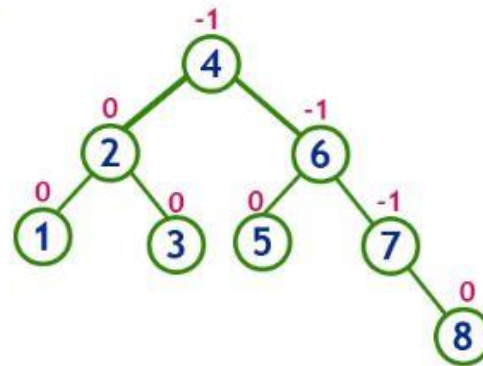
insert 7



After LL Rotation at 5



insert 8



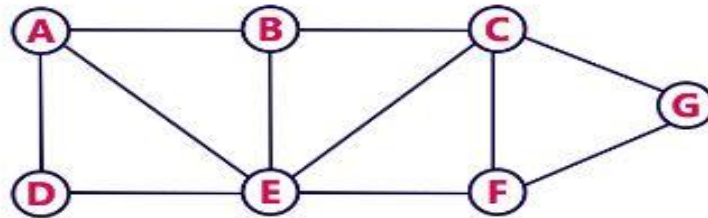
Tree is balanced

### 3. Deletion Operation in AVL Tree

The deletion operation in AVL Tree is similar to deletion operation in BST. But after every deletion operation, we need to check with the Balance Factor condition. If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.

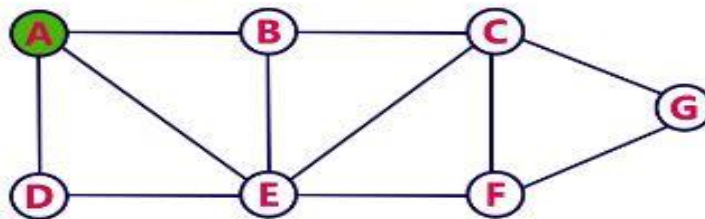
	<b>b) Demonstrate the DFS graph traversal algorithm.</b>	<b>10M</b>	<b>L3</b>	<b>C123.1</b>
	<p>DFS traversal of a graph produces a <b>spanning tree</b> as final result. <b>Spanning Tree</b> is a graph without loops. We use <b>Stack data structure</b> with maximum size of total number of vertices in the graph to implement DFS traversal.</p> <p>We use the following steps to implement DFS traversal...</p> <ul style="list-style-type: none"> <li>• <b>Step 1</b> - Define a Stack of size total number of vertices in the graph.</li> <li>• <b>Step 2</b> - Select any vertex as <b>starting point</b> for traversal. Visit that vertex and push it on to the Stack.</li> <li>• <b>Step 3</b> - Visit any one of the non-visited <b>adjacent</b> vertices of a vertex which is at the top of stack and push it on to the stack.</li> <li>• <b>Step 4</b> - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.</li> <li>• <b>Step 5</b> - When there is no new vertex to visit then use <b>back tracking</b> and pop one vertex from the stack.</li> <li>• <b>Step 6</b> - Repeat steps 3, 4 and 5 until stack becomes Empty.</li> <li>• <b>Step 7</b> - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph</li> </ul> <p><b>Back tracking</b> is coming back to the vertex from which we reached the current vertex.</p>			

Consider the following example graph to perform DFS traversal



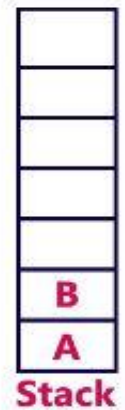
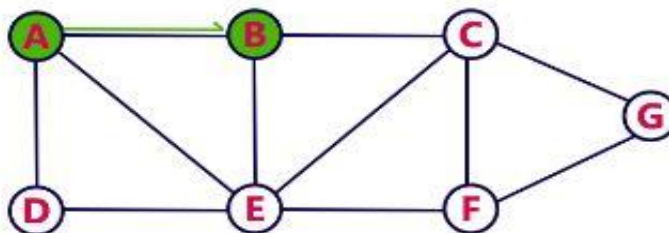
**Step 1:**

- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.



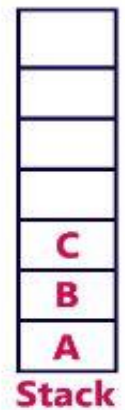
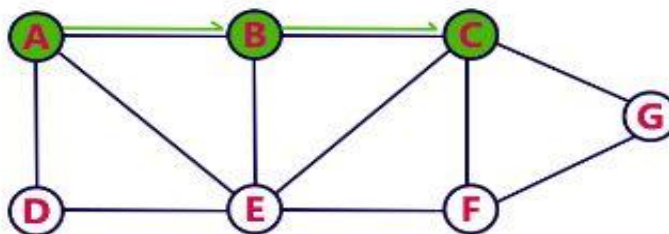
**Step 2:**

- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.



**Step 3:**

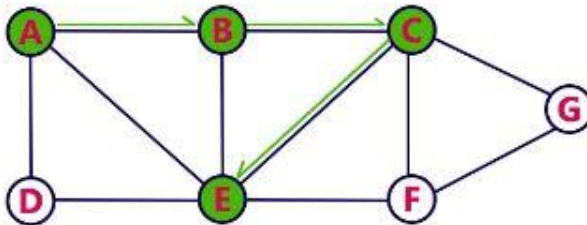
- Visit any adjacent vertex of **B** which is not visited (**C**).
- Push C on to the Stack.



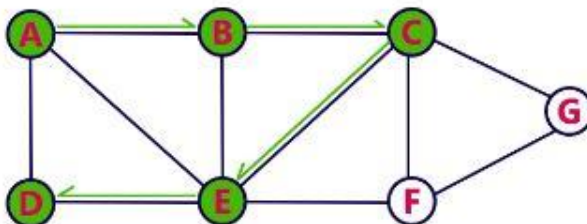


**Step 4:**

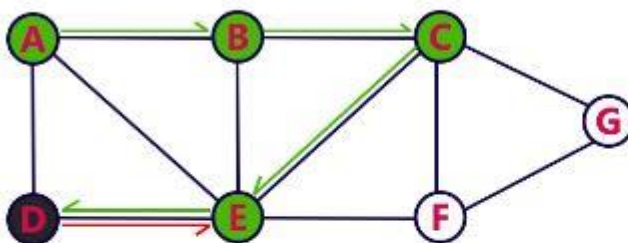
- Visit any adjacent vertex of **C** which is not visited (**E**).
- Push E on to the Stack

**Stack****Step 5:**

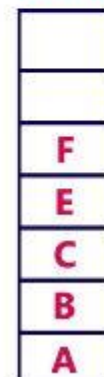
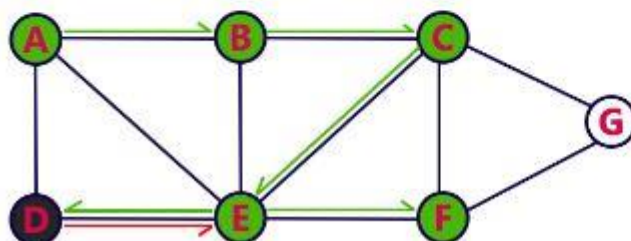
- Visit any adjacent vertex of **E** which is not visited (**D**).
- Push D on to the Stack

**Stack****Step 6:**

- There is no new vertex to be visited from D. So use back track.
- Pop D from the Stack.

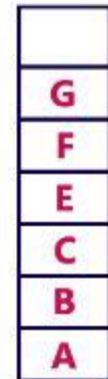
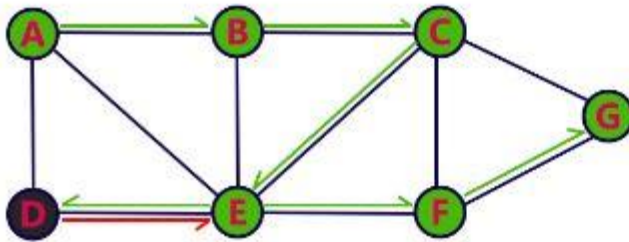
**Stack****Step 7:**

- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.

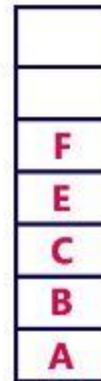
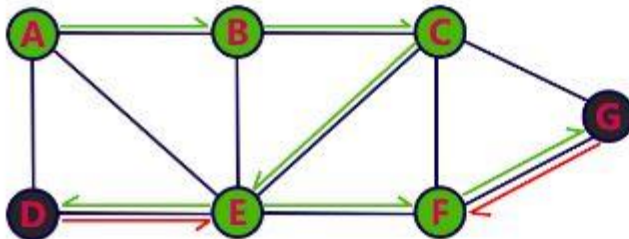
**Stack**

**Step 8:**

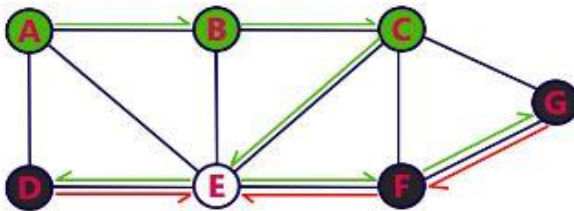
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.

**Stack****Step 9:**

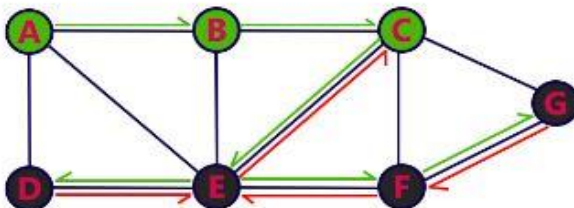
- There is no new vertex to be visited from **G**. So use back track.
- Pop **G** from the Stack.

**Stack****Step 10:**

- There is no new vertex to be visited from **F**. So use back track.
- Pop **F** from the Stack.

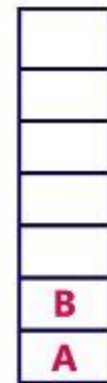
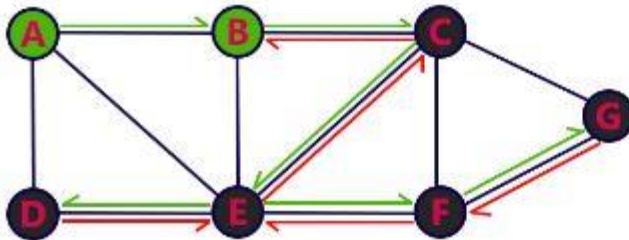
**Stack****Step 11:**

- There is no new vertex to be visited from **E**. So use back track.
- Pop **E** from the Stack.

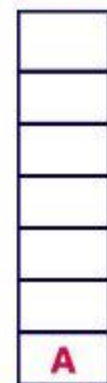
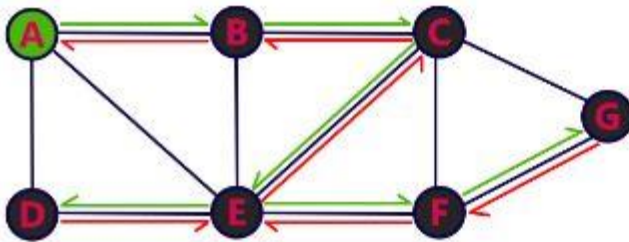
**Stack**

**Step 12:**

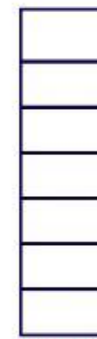
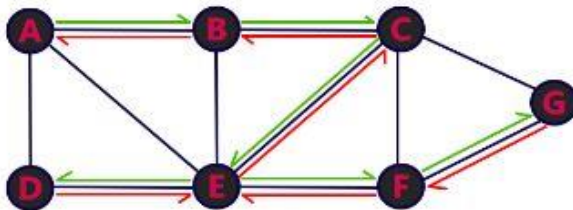
- There is no new vertex to be visited from C. So use back track.
- Pop C from the Stack.

**Step 13:**

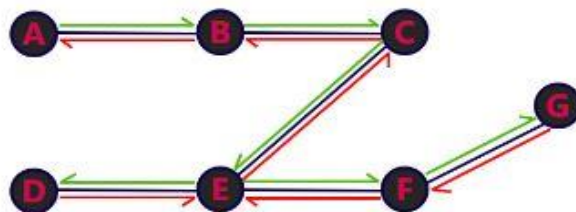
- There is no new vertex to be visited from B. So use back track.
- Pop B from the Stack.

**Step 14:**

- There is no new vertex to be visited from A. So use back track.
- Pop A from the Stack.



- Stack became Empty. So stop DFS Traversal.
- Final result of DFS traversal is following spanning tree.



## c) Discuss traversal methods in a Binary Tree.

10M

L2

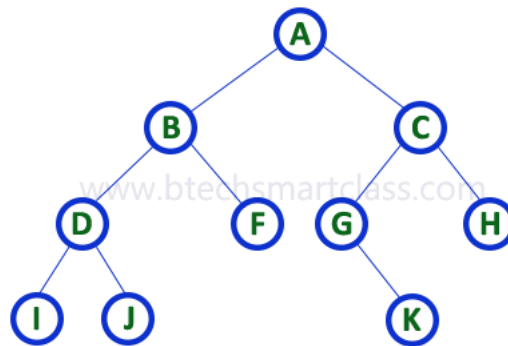
C123.1

Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

There are three types of binary tree traversals.

4. In - Order Traversal
5. Pre - Order Traversal
6. Post - Order Traversal

Consider the following binary tree...



### 1. In - Order Traversal ( leftChild - root - rightChild )

In In-Order traversal, the root node is visited between the left child and right child. 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. This in-order traversal is applicable for every root node of all subtrees in the tree. This is performed recursively for all nodes in the tree.

In the above example of a binary tree, first we try to visit left child of root node 'A', but A's left child 'B' is a root node for left subtree. so we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes I and J. So we try to visit its left child 'I' and it is the leftmost child. So first we visit 'I' then go for its root node 'D' and later we visit D's right child 'J'. With this we have completed the left part of node B. Then visit 'B' and next B's right child 'F' is visited. With this we have completed left part of node A. Then visit root node 'A'. With this we have completed left and root parts of node A. Then we go for the right part of the node A. In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit 'G' and then visit G's right child K. With this we have completed the left part of node C. Then visit root node 'C' and

next visit C's right child '**H**' which is the rightmost child in the tree. So we stop the process.

That means here we have visited in the order of **I - D - J - B - F - A - G - K - C - H** using In-Order Traversal.

**In-Order Traversal for above example of binary tree is**

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

## **2. Pre - Order Traversal ( root - leftChild - rightChild )**

In Pre-Order traversal, the root node is visited before the left child and right child nodes. In this traversal, the root node is visited first, then its left child and later its right child. This pre-order traversal is applicable for every root node of all subtrees in the tree. In the above example of binary tree, first we visit root node '**A**' then visit its left child '**B**' which is a root for D and F. So we visit B's left child '**D**' and again D is a root for I and J. So we visit D's left child '**I**' which is the leftmost child. So next we go for visiting D's right child '**J**'. With this we have completed root, left and right parts of node D and root, left parts of node B. Next visit B's right child '**F**'. With this we have completed root and left parts of node A. So we go for A's right child '**C**' which is a root node for G and H. After visiting C, we go for its left child '**G**' which is a root for node K. So next we visit left of G, but it does not have left child so we go for G's right child '**K**'. With this, we have completed node C's root and left parts. Next visit C's right child '**H**' which is the rightmost child in the tree. So we stop the process.

That means here we have visited in the order of **A-B-D-I-J-F-C-G-K-H** using Pre-Order Traversal.

**Pre-Order Traversal for above example binary tree is**

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

## **3. Post - Order Traversal ( leftChild - rightChild - root )**

In Post-Order traversal, the root node is visited after left child and right child. 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.

Here we have visited in the order of **I - J - D - F - B - K - G - H - C - A** using Post-Order Traversal.

**Post-Order Traversal for above example binary tree is**

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

