Faculty Orientation Workshop

on

Data Structures

Under the Aegis of BoS (E&TC), SPPU, Pune
SE E&TC/ Electronics) 2019 Course
(22nd to 26th June 2020)

Prof. Mrs.Rajani.P.K
Assistant Professor
Pimpri Chinchwad College of Engineering, Pune

"Challenges are what make life interesting and overcoming them is what makes life meaningful."

- **Joshua J. Marine**

Data Structures - Course Details

Programme: UG Programme in E&TC/ Electronics Engineering			Class: S.E (E&TC/ Electronics)				A.Y. 2020-21 Sem. I	
Course Code: 204184 (Theory)			Course : Data Structures					
Corresponding Lab Course Code : 204188			Lab Course Name: Data Structure Laboratory					
Teaching Scheme			Examination Scheme					
Theory	Practical	Tutorial	Theory				Lab	
(hrs/week)	(hrs/week)	(hrs/week)	Insem	Endsem	Sessional	Term Work	Practical	Oral
3 hrs	2 hrs	----	30	70	---	--	--	25

Abstract: On completion of this course, students will be familiar with C programming and should be able to implement sorting and searching algorithms and calculates it complexity. Students will be able to develop applications of stacks and queues using array and also demonstrate applicability of linear and non-linear data structures. Students will be able to design height balanced Binary Tree and apply the knowledge of graph for solving the problems of spanning tree and shortest path algorithm.

Data Structures - Course Details

Delivery Methods (DM)

Chalk & Talk	ICT Tools	Group Discussion	Industrial/ Field Visit	Expert Talk	Animation, Role Plays	Mini project	Lab
--	√	---	---	√	√	√	√

COURSE OBJECTIVES

The course aims to:

- To learn basic concepts of C Programming language
- To learn different sorting and searching algorithms and its analysis
- To learn linear data structures : Stack and Queue, Link List and applications.
- **To learn Non linear data structures : Tree , Graph and applications.**
- **To study the systematic way of solving problems, various methods of organizing large amounts of data.**
- **To solve problems using data structures such as binary trees, binary search trees, and graphs and writing programs**

Data Structures - Course Outcome

COURSE OUTCOMES

On completion of the course, learner will be able to:

CO1:Solve mathematical problems using C programming language.

CO2: Implement sorting and searching algorithms and calculate their complexity.

CO3: Develop applications of stack and queue using array.

CO4: Demonstrate applicability of Linked List.

CO5: Demonstrate applicability of nonlinear data structure–Binary Tree with respect to its time complexity.

CO6: Apply the knowledge of graph for solving the problems of spanning tree and shortest path algorithm.

Data Structures - Program Outcome

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specializatic
engineering problems.

2. Problem analysis: Identify, formulate, reviev
complex
engineering problems reaching substantiated c
mathematics, natural sciences, and engineerin

3. Design/development of solutions: Design s
problems and design system components or pr
with appropriate consideration for the public h
societal, and environmental considerations.

5. Modern tool usage: Create, select, and appl
and modern engineering and IT tools including
engineering activities with an understanding of

9. Individual and team work: Function effectiv
or leader in diverse teams, and in multidisciplinary settings.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Outcomes (POs)

1. Engineering knowledge
2. Problem analysis
3. Design/development of solutions
4. Conduct investigations of complex problems
5. Modern tool usage
6. The engineer and society
7. Environment and sustainability
8. Ethics
9. Individual and team work
10. Communication
11. Project management and finance
12. Life-long learning

Data Structures: CO-PO Mapping

Course Outcome	Blooms Taxonomy Level	After successful completion of the course students will be able to	Mapping with Syllabus Unit	PO MAPPING
CO204184.5	3	**Demonstrate** applicability of nonlinear data structure: Binary Tree with respect to its time complexity.	5	1,2,3,5,9,12

Data Structures: CO-PO Mapping...

MAPPING	LEVEL	JUSTIFICATION
CO5-PO1	1	**Apply the knowledge of mathematics**, science, engineering fundamentals for solving **Binary Tree non-linear data structure Problems**
CO5-PO2	1	**Identify, formulate and analyze** engineering problems using the concepts **Binary Tree** non-linear data structures.
CO5-PO3	1	**Design solutions for engineering problems** using non-linear data structure **Binary Tree.**
CO5-PO5	1	Select and apply **appropriate IT tools** for modeling the applications of **Binary Search Tree** using non-linear data structures.
CO5-PO9	1	Function effectively as an **individual and as a member** and **apply appropriate IT tools for modeling** the applications of **Binary Tree** using non-linear data structures.
CO5-PO12	1	**Recognize the need for Binary Tree** non-linear data structures, and have the **preparation for technological change**.

Data Structures: Books

TEXT BOOKS
T1: Ellis Horowitz, Sartaj Sahni, “Fundamentals of Data Structures”, Galgotia Books Source. T2: Richard. F. Gilberg, Behrouz A. Forouzan, “Data Structures: A Pseudocode Approach with C,” Cengage Learning, second edition.
REFERENCE BOOKS
R1: Seymour Lipschutz, “Data Structure with C, Schaum’s Outlines”, Tata McGrawHill. R2: E Balgurusamy, “Programming in ANSI C”, Tata McGraw-Hill, Third Edition. R3: Yedidyah Langsam, Moshe J Augenstein, Aaron M Tenenbaum “Data structures using C and C++” PHI Publications, 2nd Edition. R4: Reema Thareja, “Data Structures using C”, Second Edition, Oxford University Press, 2014
ADDITIONAL MATERIAL
MOOC / NPTEL: 1. NPTEL Course “Programming & Data Structure” https://nptel.ac.in/courses/106/105/106105085/ 2. NPTEL Course “Data Structure & Algorithms” https://nptel.ac.in/courses/106/102/106102064/

Extra Books

Text Books:

- 1. Yashavant Kanetkar, “Data Structures Through C”, BPB Publication, 2nd Edition(CD available)
- 2. ISRD Group,“Data Structures using C” , Mc Graw Hill

Objective Type

- 3. Yashavant Kanetkar, ”Test your C Skills”, BPB Publication.
- 4. Yashavant Kanetkar, ”Understanding Pointers in C ”, BPB Publication.

Data Structures: Topic Mapping

Sr. No.	Topic	Reference / Text book with page no.
	UNIT V: TREE	
5.1	**Introduction to trees:** Basic Tree Concepts **Binary Trees:** • Concept & Terminologies • Representation of Binary Tree in memory • Traversing a binary tree	T2(6.1-6.2) R1(7.1-7.4)
5.2	**Binary Search Trees (BST):** • Basic Concepts • BST operations • Concept of Threaded Binary Search Tree	T2(7.1-7.2,7.5) R1(7.7-7.9)
	. **AVL Tree:** Basic concepts and rotations of a Tree.	T2(8.1-8.3) R1(7.10-7.12)

•T2. Richard F. Gilberg& Behrouz A. Forouzan, Data Structures A Pseudocode Approach with C, Cengage Learning, second edition. ISBN-10: 0534390803

•R1.Seymour Lipschutz, Data Structure with C, Schaum's Outlines, Tata McGrawHill.ISBN-10: 1259029964

Data Structures: Lab Experiment

Group A (Compulsory)	
Write a C program to:	
1	Perform following String operations with and without pointers to arrays (without using the library functions): a. substring b. palindrome c. compare d. copy e. reverse
2	Implement Database Management using array of structures with operations Create, Display, Modify, Append, Search and Sort. (For any database like Employee or Bank database with and without pointers to structures)
3	Implement Stack and Queue using arrays.
4	Create a singly linked list with options: a. Insert (at front, at end, in the middle), b. Delete (at front, at end, in the middle), c. Display, d. Display Reverse, e. Revert the SLL
5	**Implement Binary search tree with operations Create, search, and recursive traversals.**
6	Implement Graph using adjacency Matrix with BFS & DFS traversals.

Data Structures: Lab Experiments

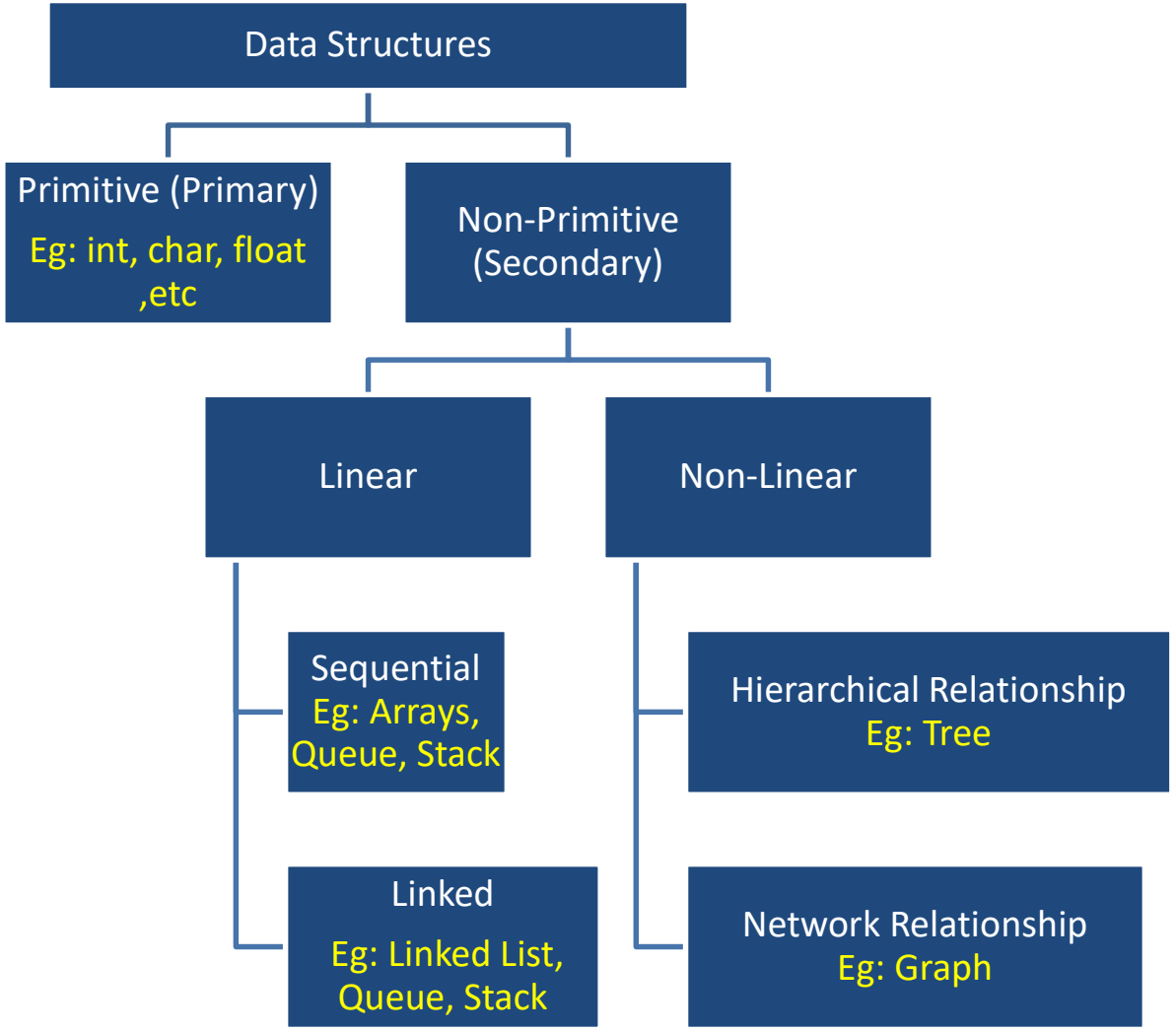
Group B :(Perform any 3)	
Write a C program to:	
1	7.Implement stack and queue using linked list.
2	8.Implement assignment 2 using files.
3	9.Add two polynomials using linked list.
4	10.Reverse a doubly linked list.
5	11.Evaluate postfix expression (input will be postfix expression)
6	12.Reverse and Sort stack using recursion.
7	13.Implement inorder tree traversal without recursion.
8	14.To find inorder predecessor and successor of a given key in BST.
9	15.Implement Quicksort.

Data Structures: Lab Experiments

Group C: (Perform any 1) Write a C program to:	
1	16. Implement merge sort for doubly linked list.
2	17. Construct a tree from given inorder and preorder traversal.
3	18. Implement Dijkstra's Algorithm.
4	19.Implement Circular Linked List with various operations.
5	20. Represent graph using adjacency list or matrix and generate minimum spanning tree using Prism's algorithm.

Classification of Data Structure

https://www.youtube.com/watch?v=d_XvFOkQz5k



Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

Trees- Concept

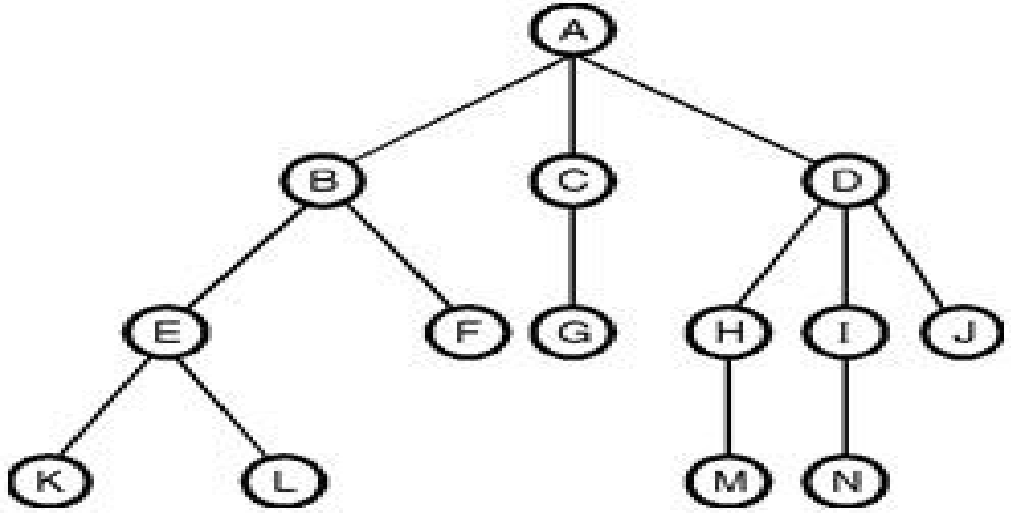
- Linear DS- arrays, stack ,queue, LL
- Nonlinear DS-Trees ,Graphs
- Trees

(https://www.youtube.com/watch?v=ikPPdBDZnz4&t=149s)

  - Used to represent the data objects in hierarchical manner.
  - Represents some kind of relationship between the nodes of the tree.
  - Every particular node may have leaf node or parent node.
  - In computer science, the tree DS concept is leaves are below and root is at the top.(just opposite as normal tree)
- Definition of Tree
- A tree is finite set of one or more nodes such that:
  - There is a specially designated node called as root.
  - The remaining nodes are called as subtrees of the root.

Tree Terminology

- **Root node:** Unique node in the tree to which further sub trees are attached. [A]
- **Leaf nodes (leaves):**terminal nodes of the tree[K,L,F,G,M,N]
- **Degree of tree** : The total number of subtrees of a node is called as its **degree**. [The degree of A is 3, that of C is 1.]
- **Level of the node is its distance from the root.** [Level is 3]
- **Height or depth of tree:** It is the maximum level of any node in the tree. [The maximum level Tree shown is 4, and thus the height of the tree is 4.]
- **The height of the tree =level of the leaf in the longest path from root +1**
- **Height of empty Tree is -1**



Why Tree?

- Unlike Array and Linked List, which are linear data structures, tree is hierarchical (or non-linear) data structure.

1. One reason to use trees might be because you want to store information that naturally forms a hierarchy.

2. If we organize keys in form of a tree (with some ordering e.g., BST), we can search for a given key in moderate time (quicker than Linked List and slower than arrays). Self-balancing search trees like AVL guarantee an upper bound of O(Logn) for search.

3. We can insert/delete keys in moderate time (quicker than Arrays and slower than Unordered Linked Lists). Self-balancing search trees like AVL guarantee an upper bound of O(Logn) for insertion/deletion.

4. Like Linked Lists and unlike Arrays, Pointer implementation of trees don't have an upper limit on number of nodes as nodes are linked using pointers.

Applications of Tree

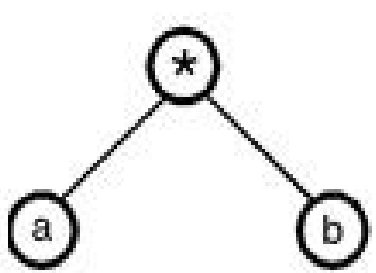
1. Store hierarchical data, like folder structure, organization structure, XML/HTML data.
2. Binary Search Tree is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
3. Spanning Trees and shortest path trees are used in routers and bridges respectively in computer networks
4. Syntax Tree: Used in Compilers.
5. K-D Tree: A space partitioning tree used to organize points in K dimensional space.
6. Trie : Used to implement dictionaries with prefix lookup.
7. Suffix Tree : For quick pattern searching in a fixed text.
8. As a workflow for compositing digital images for visual effects.
9. Expression Tree: applications of BT

Decision Trees / Classification Trees

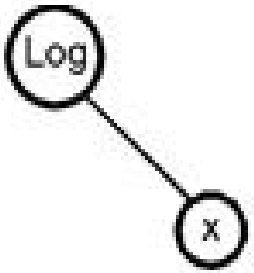
- A *Decision Tree*, more properly a *classification tree*, is used to learn a *classification function* which predicts the value of a *dependent attribute* (variable) given the values of the independent (input) attributes (variables).
- This solves a problem known as *supervised classification* because the dependent attribute and the number of classes (values) that it may have are given.
- Binary Attributes

	@1 petrol in tank?	@2 headlights work?	@3 engine turns over	@4 electrical problem
case 1:	y	y	n	y
case 2:	n	y	y	n
case 3:	y	n	n	n
... ... etc.				
- **Cases** -				

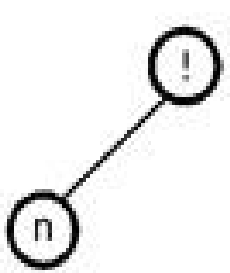
Expression Tree



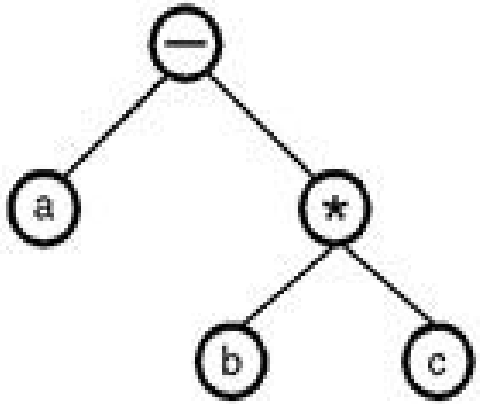
*a * b*



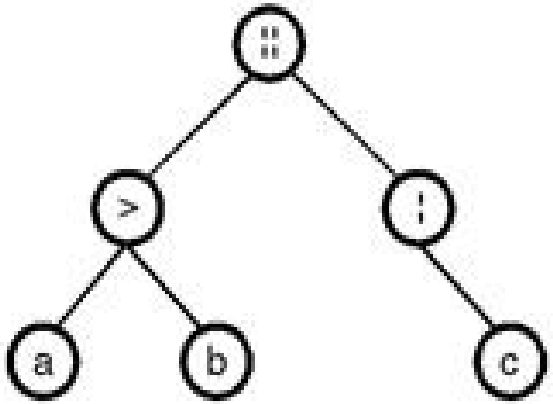
Log x



n! (factorial n)



*a – b * c*



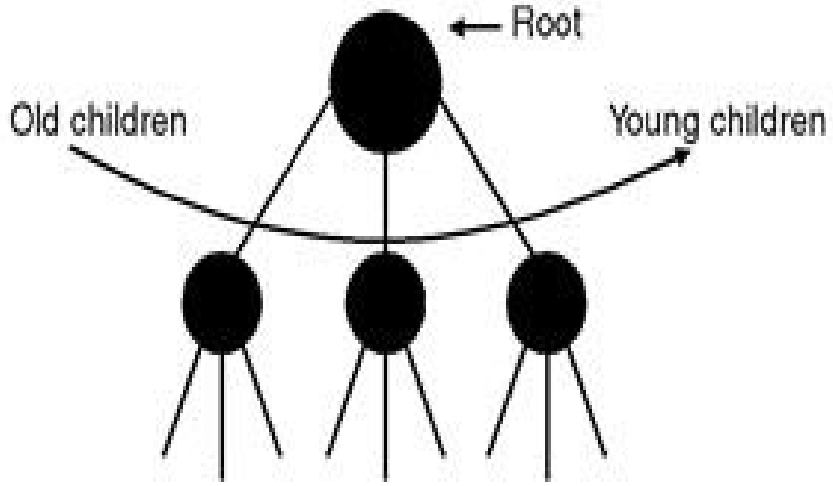
(a > b) :: (!c)

Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

General Tree

- Ordered tree- An ordered tree consists of a root node and any number of children which are ordered from "oldest" to "youngest", hence the name.



Binary tree

- A binary tree is a finite set of nodes is either empty or with root and two disjointbinary trees called left subtree and right subtree.
- https://www.youtube.com/watch?v=7vw2iIdqHlM&t=21s

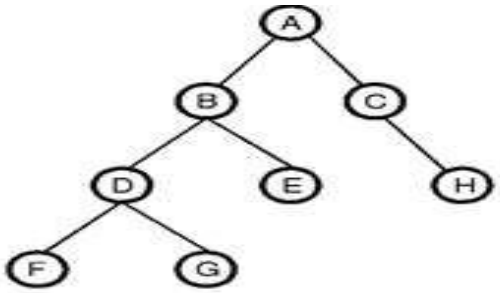


Fig. 1.10


Fig. 1.11

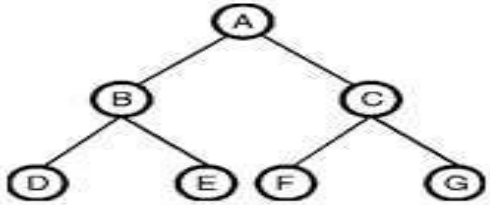


Fig. 1.12

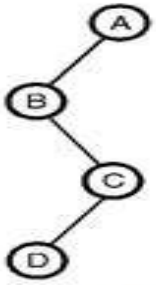


Fig. 1.13

Unit V: Trees

Complete Binary Tree

- A complete binary tree is defined as a binary tree where

(i) All of its leaves are on level n – 1 or level n.

(ii) On levels 1 through n – 2, every node has two children.

(iii) On level n the leaves are as far to the left as possible

A BT of depth n having 2^n-1 maximum node

Eg: depth, n=3 …..>>maximum nodes=7

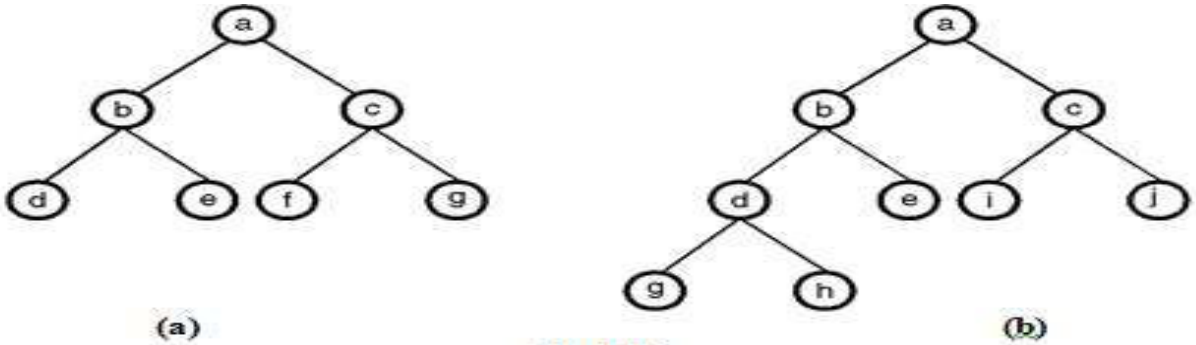


Fig. 1.16

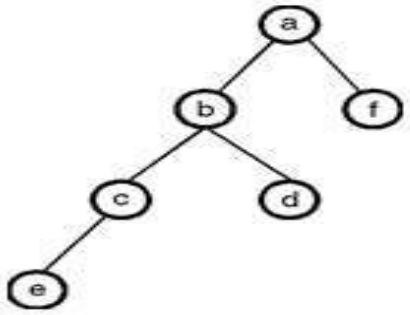


Fig. 1.17

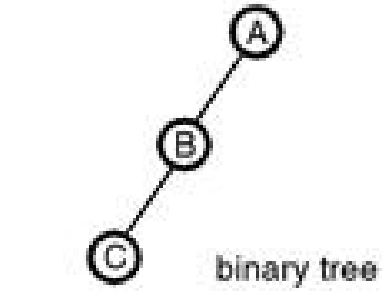
Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

Representation of Binary Tree in memory

Tree representation using:

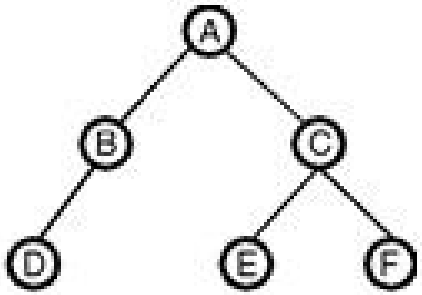
- Sequential (Array)
- Linked List



binary tree

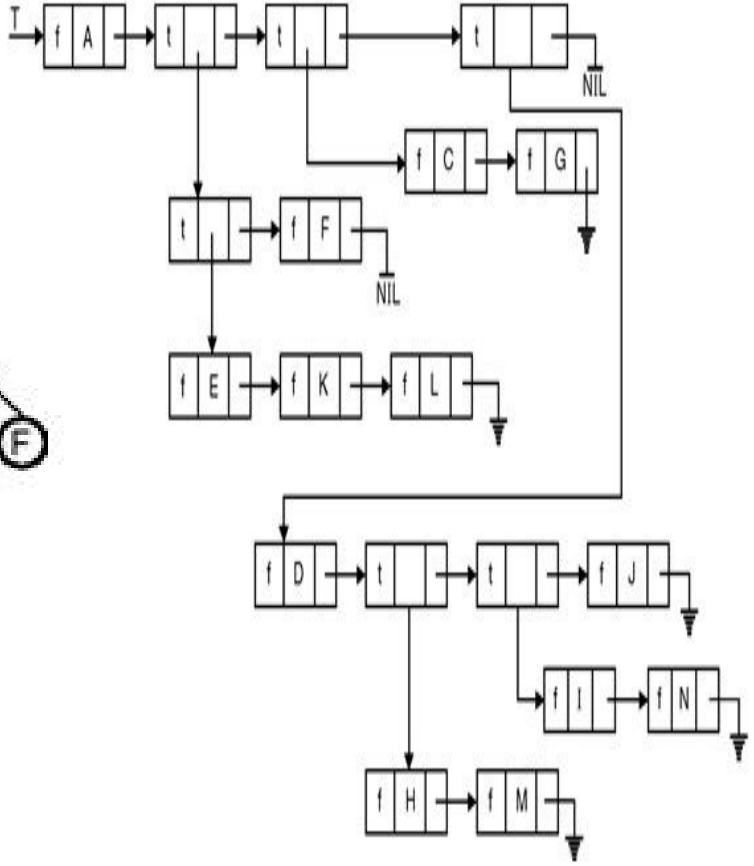
Array representation :

1	A
2	B
3	—
4	C
5	—



Array representation :

A
B
C
D
—
E
F



Sequential Representation Binary Tree

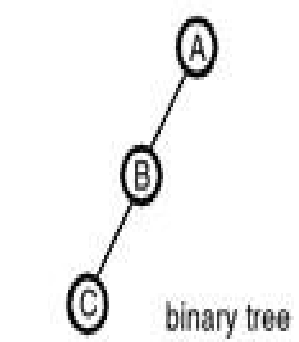
- Array representation of binary tree

A binary tree of depth n having 2^n-1 maximum node

Eg: depth, n=3 …..>>maximum nodes=7

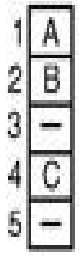
Root node index=0, left=2n+1, right=2n+2

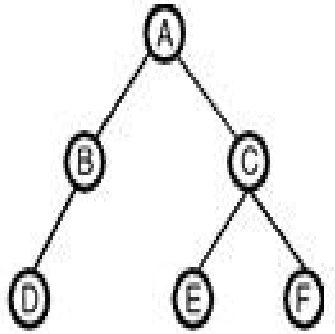
[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	-	C	-	-	-



binary tree

Array representation :





Array representation :

[0]	[1]	[2]	[3]	[4]	[5]	[6]
A	B	C	D	-	E	F

Sequential Representation Binary Tree

Advantage:

Direct Access to any node can be possible finding the parent or left/right child of any particular node is fast because of random access.

Disadvantage:

1.Wastage of memory

Eg:If half of the tree is unutilized.

2.Max. depth of the tree is fixed, because the array size is fixed.

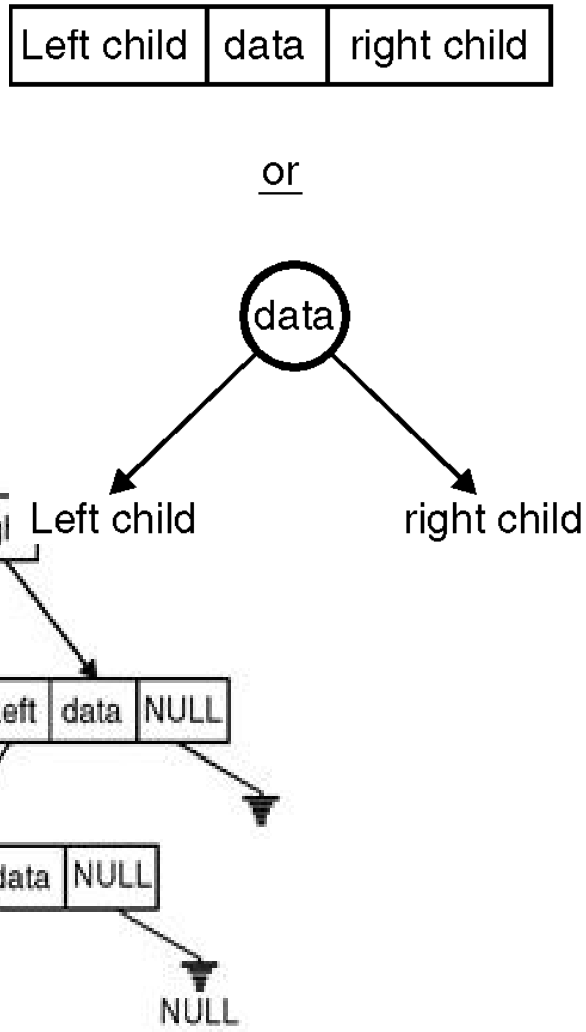
3.Insertion and deletion of any node in the tree will be costlier as the node has to be adjusted at the appropriate positions so that meaning of the binary tree can be preserved.

So more flexible representation in which linked list to represent the tree instead of array.

Linked presentation of Binary Tree

- Advantages and drawbacks
- Structure declaration of Linked BT

```
typedef struct node
{
struct node *left;
int data;
struct node *right;
}bintree;
```



Linked Representation of BT

Advantage:

1.Superior to array as there is no wastage of memory. So need to have prior knowledge of depth of the Tree. Using the concept of dynamic memory allocation, one can create as much nodes as required. At the same time it can delete the nodes by making the address free.

2.Insertions and Deletions which are common operations can be done without moving the other nodes.

Disadvantage:

1.This does not provide direct access to a node and special algorithms are required.

2.This needs additional space in each node for storing at the left and right subtrees.

Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

Tree traversals

- A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once
- Tree traversals are naturally recursive
- Since a binary tree has three “parts,” there are six possible ways to traverse the binary tree.But from computing point of 3 different ways traversing a tree.
  - root, left, right
  - left, root, right
  - left, right, root
  - root, right, left
  - right, root, left
  - right, left, root

Algorithms For Binary Tree Traversal

- Traverse operation on the tree :
  - so as to visit the each node, exactly once.
- A complete traversal gives the linear ordering of the information in the tree, may be very much useful.
- These types have been classified on the basis of, from where we get the information first.
- Left=L Right=R Root/Parent =D
- The traversal types are :

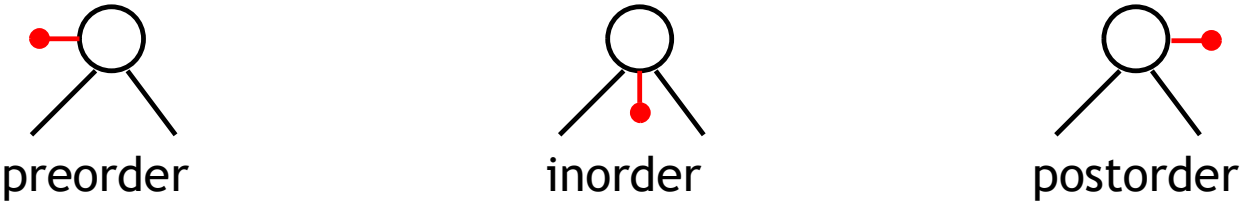
(i) Preorder traversal(DLR-root, left, right)

(ii) Inorder traversal (LDR- left, root, right)

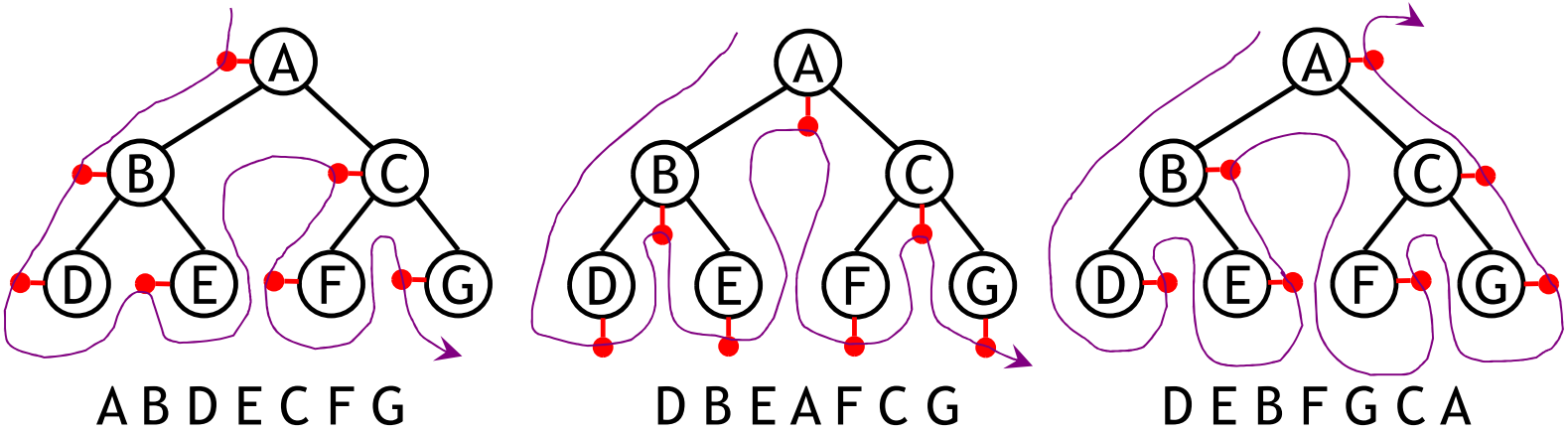
(iii) Postorder traversal (LRD- left, right, root)

Tree traversals using “flags”

- The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “flag” attached to each node, as follows:



- To traverse the tree, collect the flags:

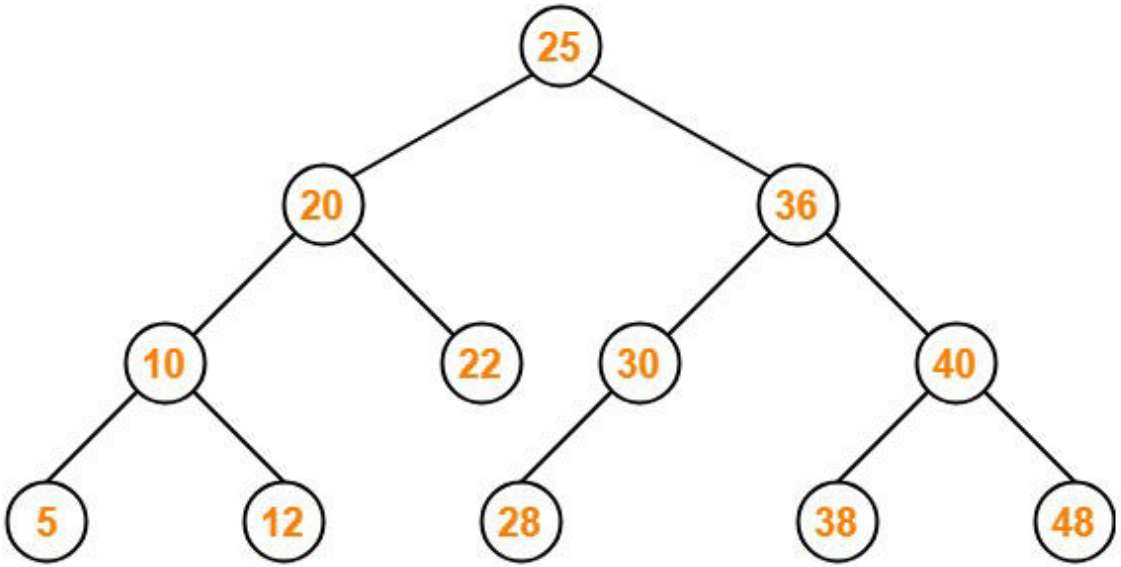


Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

Binary Search Tree (BST)

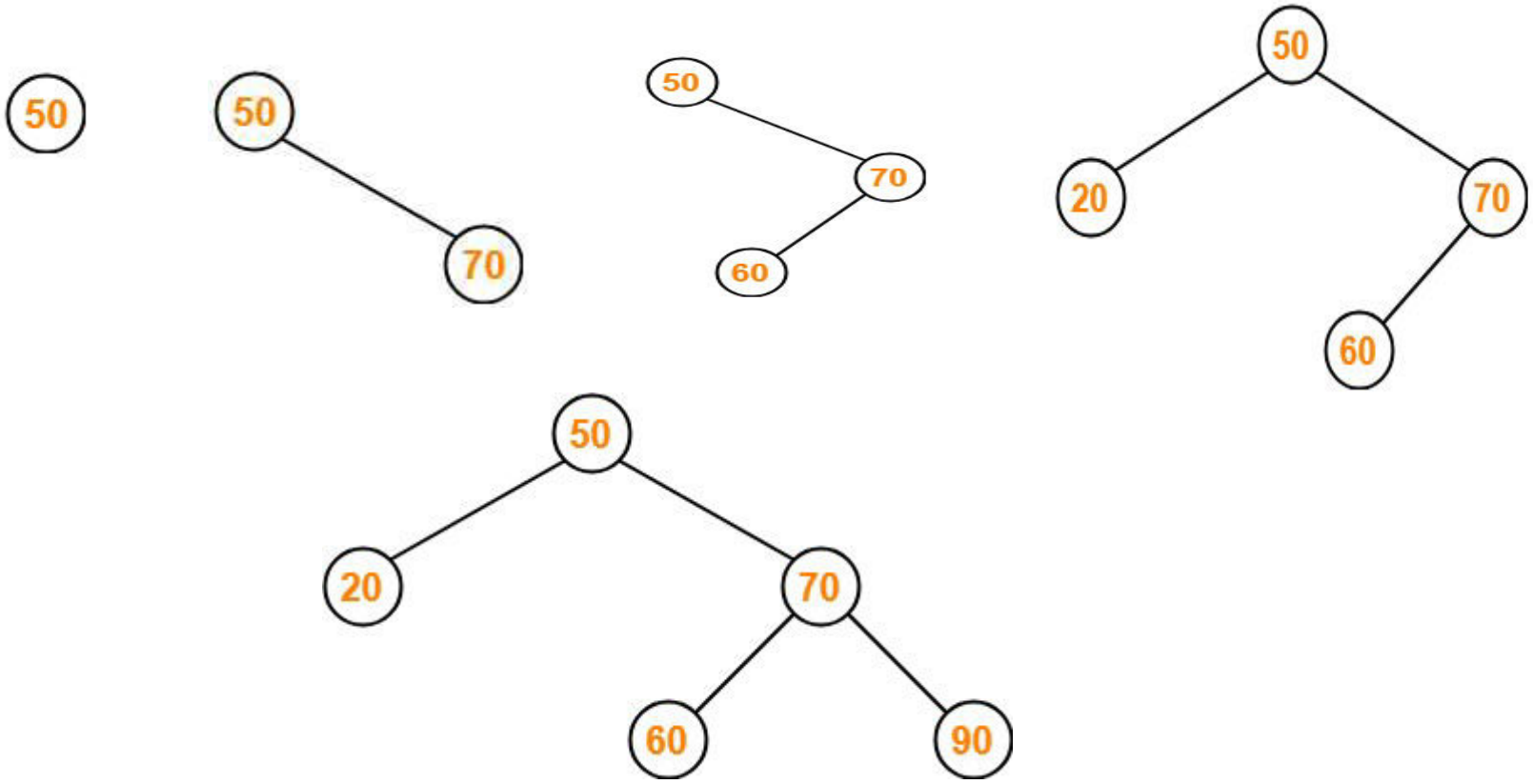
- BST
- Binary Trees are arranged in some fashion so as to make searching faster.
- **Values of left subtree < Root value < Values of right subtree**



Binary Search Tree

Binary Search Tree (BST)…

- Construct a Binary Search Tree (BST) for the following sequence of numbers:50, 70, 60, 20, 90



Binary Search Tree (BST)…

- Difference between :
  - BT :Nodes can be arranged in any manner.
  - BST: To make the searching algorithm faster in a BT
    - **Values of left subtree < Root value < Values of right subtree**
- Eg:

  1)10,7,5,9,15,12,18

  2)5,2,8,4,1,9

  3)100,50,200,300,20,150,70,180,120,30 (10 NODES)

  4) JAN,FEB,MARCH,APRL

  5) JAN,FEB,MARCH,APRL,OCT,JUN,JUL,AUG,SEP,NOV,DEC (11)

A tree can be viewed as a *recursive data structure*. Why?
Remember that *recursive* means that something is defined in terms of *itself*.
Here, this means that trees are made up of *subtrees*.

Recursive Inorder algorithm

The inorder traversal gives the sequence : 10,5,20

- The **algorithm** for inorder traversal is :

(Let ‘T’ print to the root)

Step 1 : Save ‘T’ onto stack.

Step 2 : Move to the left, as far as possible. (While doing so, save address of T on stack).

Step 3 : If now left of ‘T’ is NULL, then display the information at T.

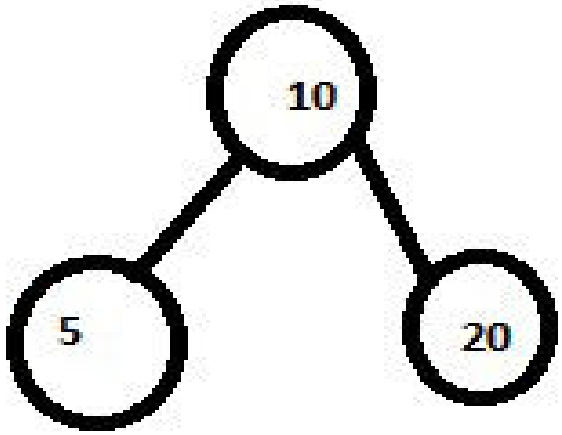
Step 4 : Move to the right. If it is NULL then take out ‘T’ from stack

Step 5 : If stack is not empty goto step 2

Step 6 : Stop.

Recursive Inorder Traversal

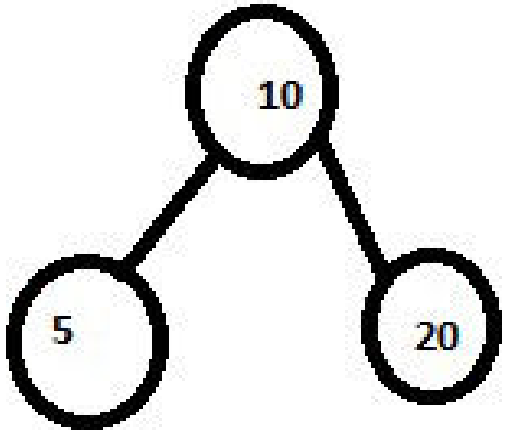
```
/* Reursive Inorder Traversal[LDR] Function */
void inorder(BSTnode *T)
{
    if(T!=NULL)
    {
        inorder(T->left);
        printf("%d\t",T->data);
        inorder(T->right);
    }
}
```



Output: 5,10,20

Recursive Preorder Traversal

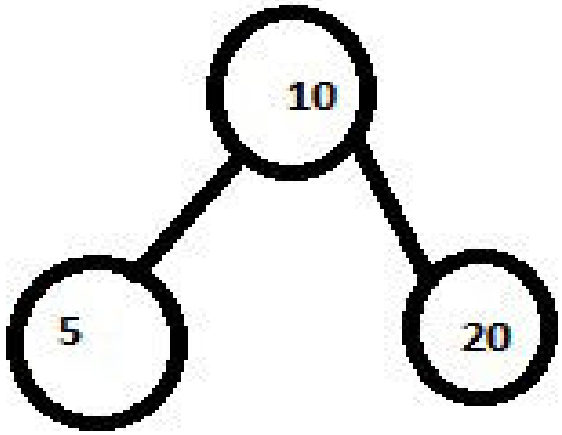
```
/* Reursive Preorder Traversal[DLR] Function */
void preorder(BSTnode *T)
 {      if(T!=NULL)
     { printf("%d\t", T-> data);
       preorder (T-> left);
       preorder (T->right);
     }
 }
```



Output: 10,5,20

Recursive Preorder Traversal

```
/* Reursive Postorder Traversal[LRD] Function */
void postorder(BSTnode *T)
 {       if(T!=NULL)
     {
       postorder (T->left);
       postorder (T->right);
       printf("%d\t",T-> data);
     }
 }
```



Output: 5,20,10

Output of BST Operations & Traversal

```
/************************OUTPUT*****************************************

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder     6)Postorder     7)Quit
Enter Your Choice :1
Enter the total number of nodes :3
Enter tree values :10 5 20

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder  6)Postorder     7)Quit
Enter Your Choice :2
Enter the key to be searched :5
  Found

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder     6)Postorder     7)Quit
Enter Your Choice :4
5         10      20

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder  6)Postorder     7)Quit
Enter Your Choice :5
10        5       20

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder     6)Postorder     7)Quit
Enter Your Choice :6
5         20      10

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder   6)Postorder     7)Quit
Enter Your Choice :7
```

Output of BST Operations & Traversal

```
**********************************************************************
1)Create      2)Search      3)Insert      4)Inorder      5)Preorder   6)Postorder    7)Quit
Enter Your Choice :1
 Enter the total number of nodes :10
Enter tree values :100 50 200 300 20 150 70 180 120 30

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder  6)Postorder    7)Quit
Enter Your Choice :4
20      30      50      70      100     120     150     180     200     300

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder   6)Postorder    7)Quit
Enter Your Choice :5
100     50      20      30      70      200     150     120     180     300

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder  6)Postorder    7)Quit
Enter Your Choice :6
30      20      70      50      120     180     150     300     200     100

1)Create      2)Search      3)Insert      4)Inorder      5)Preorder   6)Postorder    7)Quit
Enter Your Choice :7
**********************************************************************/
```

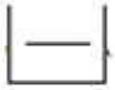
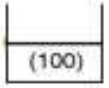
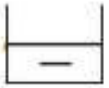
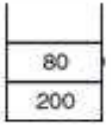
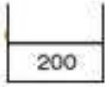
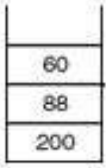
Non-recursive Traversal Algorithm

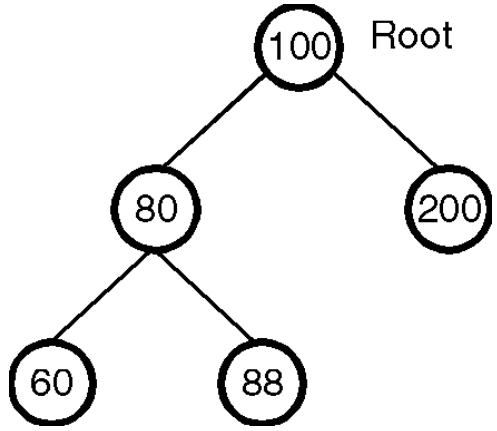
Non-recursive Method for Preorder Traversal of Binary Tree

```
void preorder_tr()
{
  // let root be the pointer to the beginning of tree
push(root);
 while (stack  is not empty)
 {
   current_node = pop();
  print the information at current_node;

  if (current_node -> right != null)
        push(current_node->right);
  if (current_node ->left != null)
          push(current_node->left);
 }
                                                    }
```

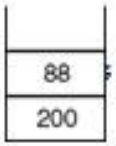
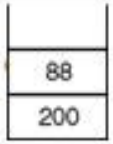
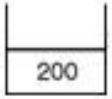
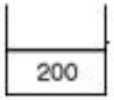
Non-recursive Method for Preorde Traversal (DLR)

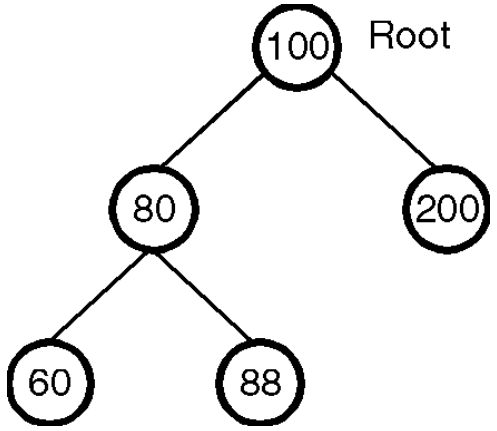
Step	Stack (Contents are addresses of node)	Remarks
0		–
1		push (root) is address of root
2		Current_node = pop( ) = (100) **Print 100**
3		Since current node's right and left are not null, push both the right is left nodes onto stack.
4		Stack is not empty, so current_node = pop( ) = 80 **Print this, 80**
5		Reason : Same as mentioned in step 3.



100,80,60,88,200

Non-recursive Method for Preorder Traversal (DLR)…

Step	Stack (Contents are addresses of node)	Remarks
6		Stack not empty, Current_node = pop( ) = 60 **Print 60**
7		Nothing is pushed onto stack, as 60's both the links are NULL.
8		Current_node = pop( ) = 88 **Print 88**
9		Reason same as in step 7, as 88 has NULL links.
10		Current_node = pop( ) = 200 **Print 200**
11		Stack is empty, so stop.



100,80,60,88,200

Non-recursive Traversal Algorithm

Non-recursive Method for Inorder Traversal of Binary Tree **:**

Step 1 : Initialize the necessary variables.
(Stack top = 0, ‘p’ pointing to the root of tree).

Step 2 : As long as ‘p’ is not NULL.
(move towards left),
top = ;
push ‘p’ onto top of stack.
move p to its left.

Step 3 : If stack is not null then
- get the node ‘p’ from top of stack;
- decrement the stack pointer.
- Display the data part of ‘p’.
else goto step 5.

Step 4 : Move ‘p’ to its right.
and goto step 2.

Step 5 : Stop

Non-recursive Traversal Algorithms

- Non-recursive Method for Postorder Traversal of Binary Tree

```
void post_order()
{
        // let root be the pointer to the beginning of tree
        // let q be a pointer
                q=root;
        flag=0; // will be set to 1 once the all elements of tree are visited.
        push(q);
         while (q !=NULL)
        {
        q= pop(); // get back address and save it as well
        push(q);
        if (q -> right != null)
        push(q->right);
        if (q ->left != null)
            push(q->left);
        if (q->left ==NULL && q->right ==NULL)
        {
                // finished with left part of the tree
        q=pop();
}
```

Non-recursive Traversal Algorithm

```
while(root->right != q) || (q!=root)
  {
//as long as we wont get the right node of root that has been pushed onto stack, as it becomes next tree for visit or if it does not have the right subtree. (remember this w.r.t root)
            print (q->info);
            q=pop();
  }
  if (q!=root) // right sub tree exists
  push(q);
 }
else
  {
   print q->info;   // root element
   flag=1;
}
}
```

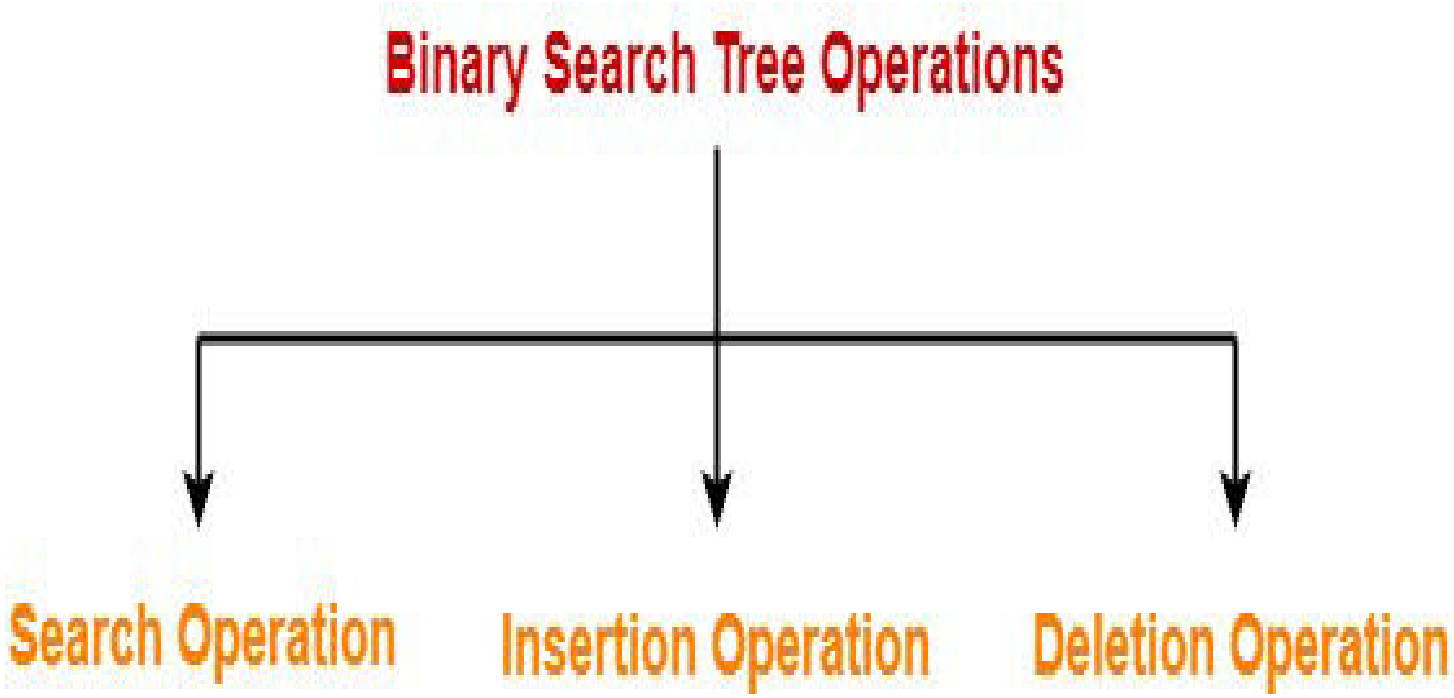
Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

Binary Search Tree Operations

Commonly performed operations on binary search tree are:

- Search Operation
- Insertion Operation
- Deletion Operation



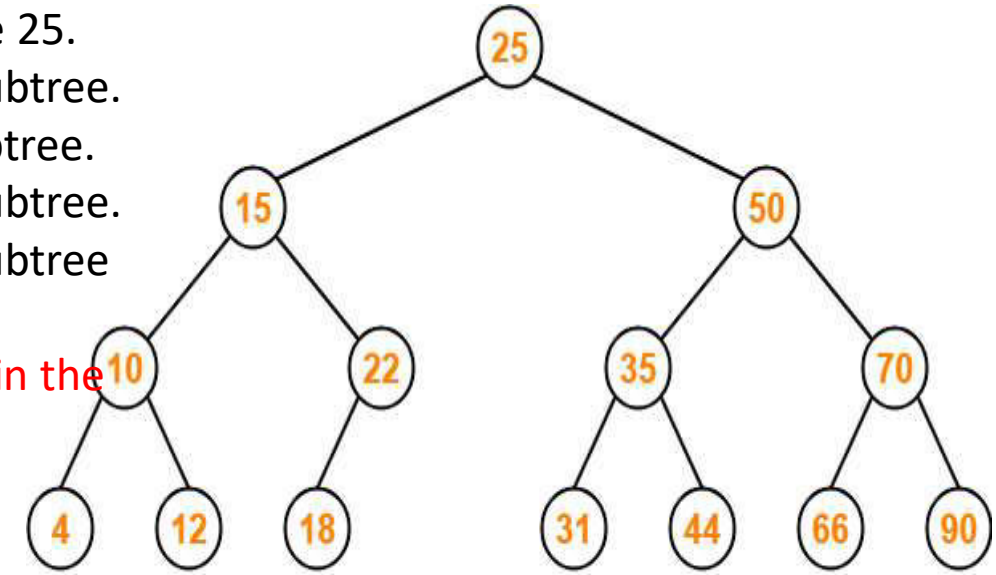
Binary Search Tree Operations-Search

Rules-

- For searching a given key in the BST,
- Compare the key with the value of root node.
- If the key is present at the root node, then return the root node.
- If the key is greater than the root node value, then recur for the root node's right subtree.
- If the key is smaller than the root node value, then recur for the root node's left subtree.

Example: Consider key = 45 has to be searched in the given BST

•We start our search from the root node 25.
•As 45 > 25, so we search in 25's right subtree.
•As 45 < 50, so we search in 50's left subtree.
•As 45 > 35, so we search in 35's right subtree.
•As 45 > 44, so we search in 44's right subtree but 44 has no subtrees.
•So, we conclude that 45 is not present in the above BST.



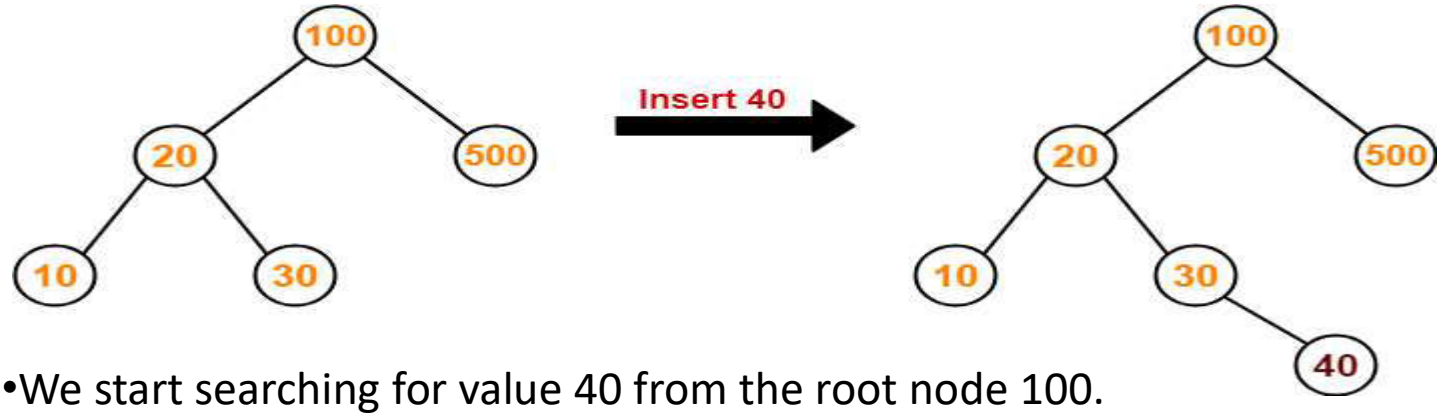
Binary Search Tree

Binary Search Tree Operations: Insertion

Rules-

- The insertion of a new key always takes place as the child of some leaf node.
- For finding out the suitable leaf node,
- Search the key to be inserted from the root node till some leaf node is reached.
- Once a leaf node is reached, insert the key as child of that leaf node.

- **Example:** Consider the following example where key = 40 is inserted in the given BST

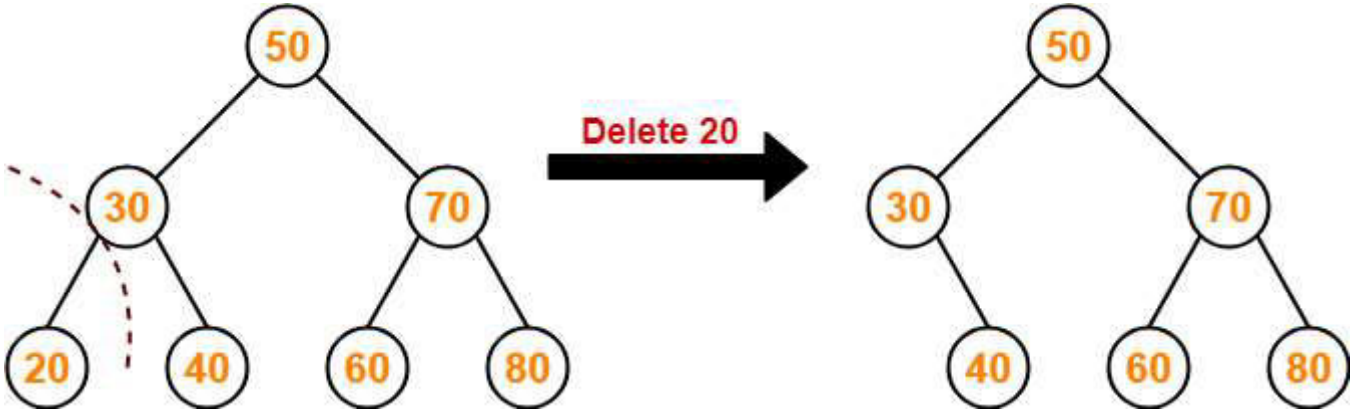


- We start searching for value 40 from the root node 100.
- As 40 < 100, so we search in 100's left subtree.
- As 40 > 20, so we search in 20's right subtree.
- As 40 > 30, so we add 40 to 30's right subtree.

Binary Search Tree Operations: Delete

- When it comes to deleting a node from the binary search tree, following three cases are possible
- **Case-01: Deletion Of A Node Having No Child (Leaf Node)**
- Just remove / disconnect the leaf node that is to deleted from the tree.

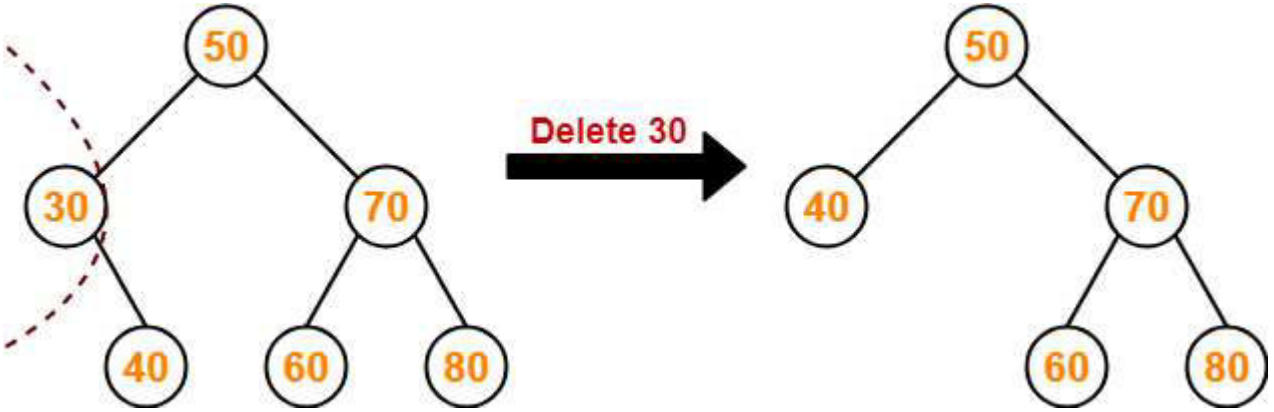
- **Example:** Consider the following example where node with value = 20 is deleted from the BST



Binary Search Tree Operations: Delete

- **Case-02: Deletion Of A Node Having Only One Child-**
- Just make the child of the deleting node, the child of its grandparent.

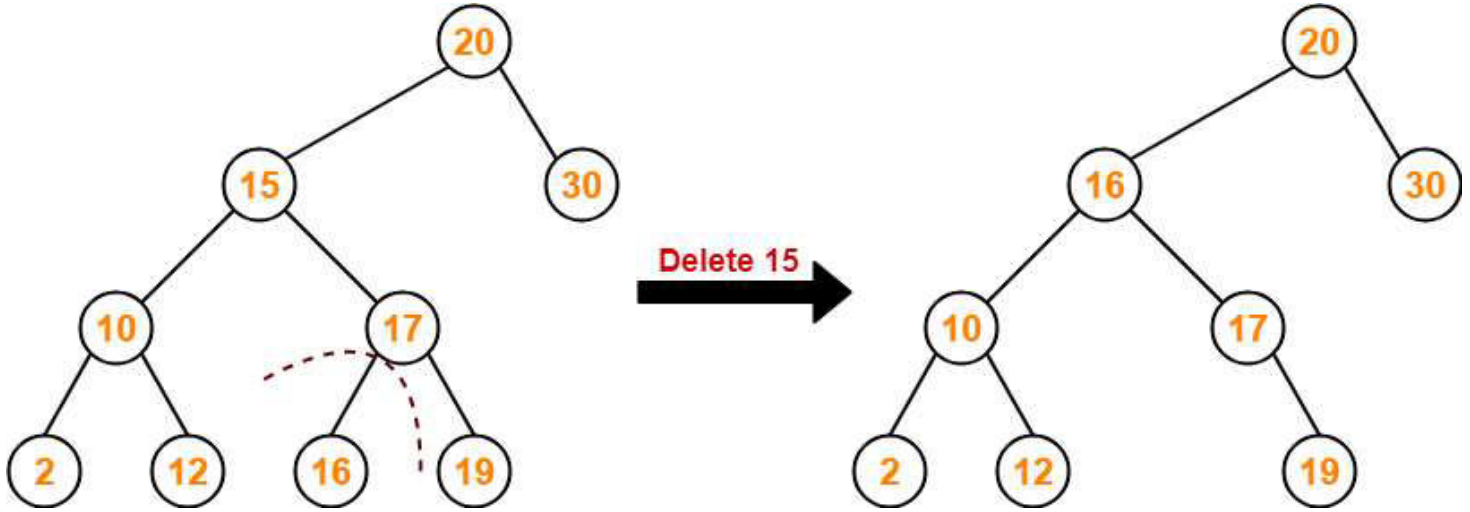
- **Example-**
- Consider the following example where node with value = 30 is deleted from the BST



Binary Search Tree Operations: Delete

- **Case-03: Deletion Of A Node Having Two Children-**
- A node with two children may be deleted from the BST in the following two ways-
-
- **Method-01:**
- Visit to the right subtree of the deleting node.
- Pluck the least value element called as inorder successor.
- Replace the deleting element with its inorder successor.

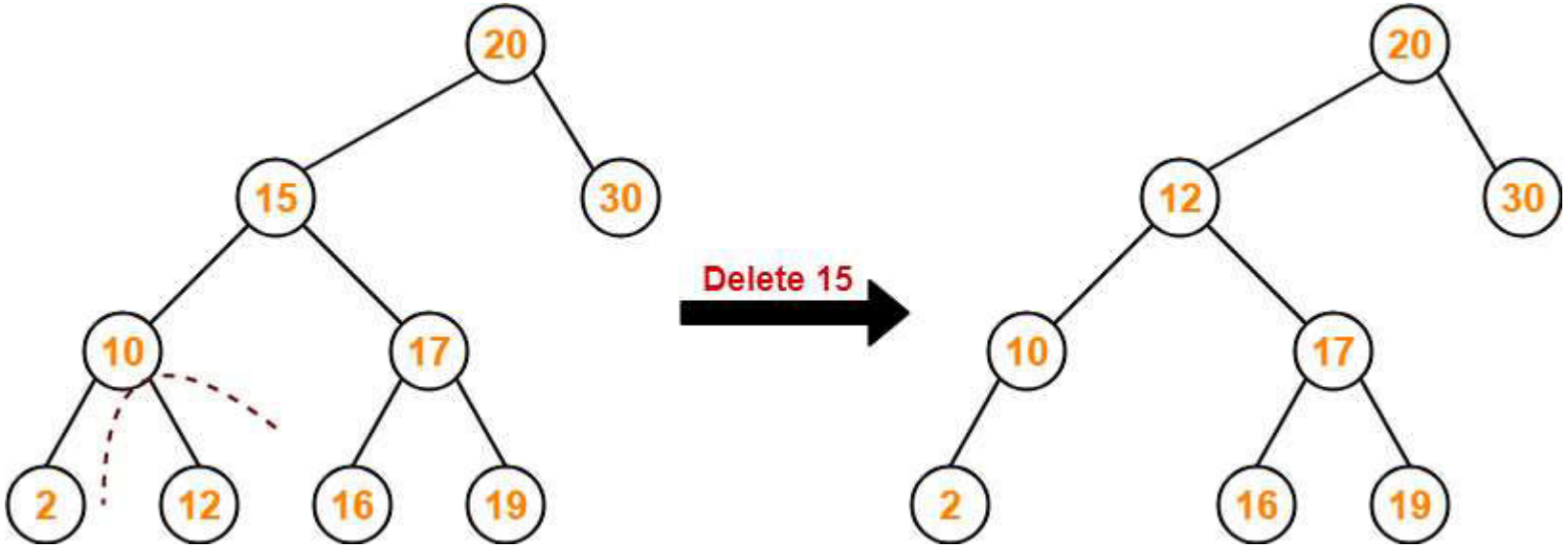
- **Example:** Consider the following example where node with value = 15 is deleted from the BST



Binary Search Tree Operations: Delete

Case-03: Deletion Of A Node Having Two Children-

- A node with two children may be deleted from the BST in the following two ways-
- **Method-02:**
- Visit to the left subtree of the deleting node.
- Pluck the greatest value element called as inorder predecessor.
- Replace the deleting element with its inorder predecessor.
- **Example:**
- Consider the following example where node with value = 15 is deleted from the BST



Unit V: Trees

Construction of a binary tree from infix and prefix expressions.

Consider the inorder/infix sequence :

a – b * c + d

and preorder sequence,

* – ab + cd

To construct the equivalent binary tree follow the steps given below :

Step 1 : Read the preorder sequence, the first element becomes the root.

Step 2 : Now scan the infix sequence, till you get an element found in step 1. Place all the elements left of this element (of infix expression) to the left of root and others to right.

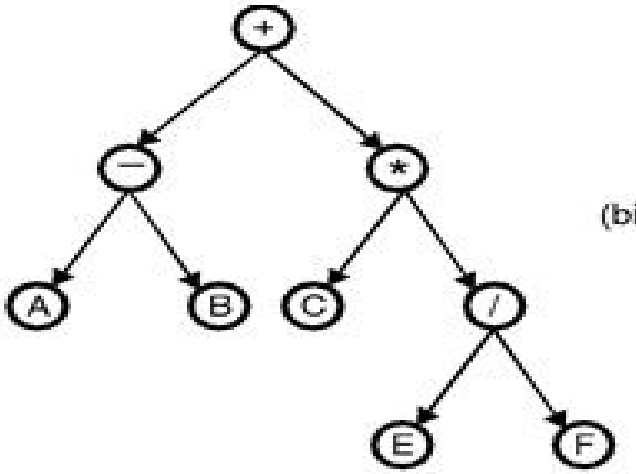
Step 3 : Repeat steps 1 and 2, till all the elements from infix sequence gets placed in tree.

Step 4 : Stop.

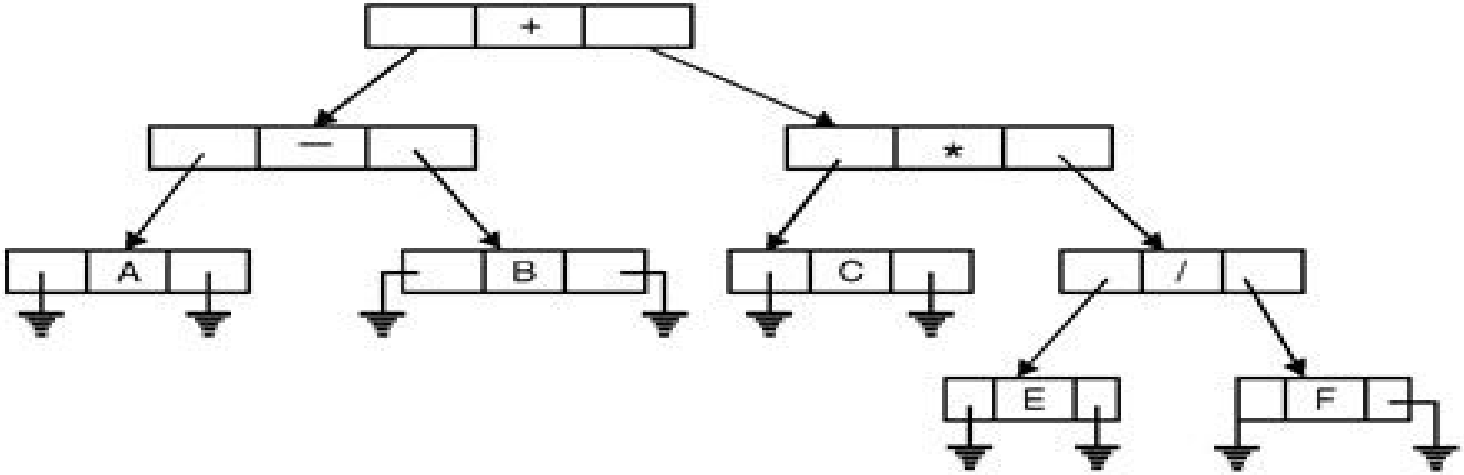
Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

Threaded Binary Trees



(binary tree)

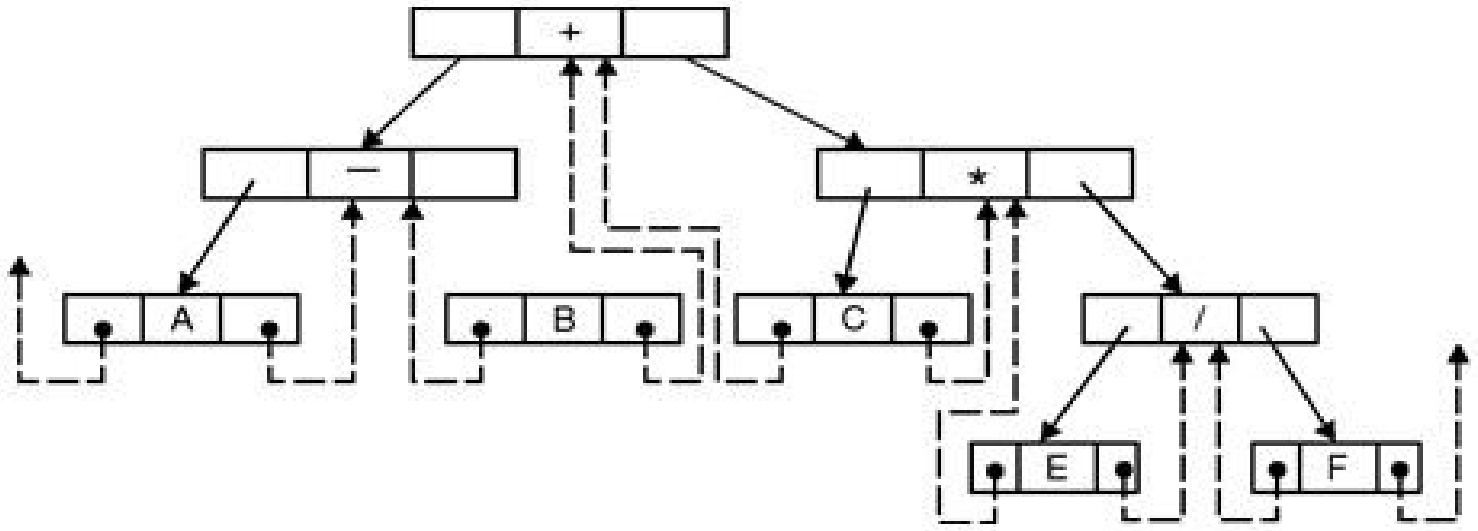


(Linked representation of above tree)

Threaded Binary Trees

- In a Linked representation of a BT, there are no more NULL Links.
- This NULL links can be replaced by pointers called threads to other nodes.
- A left NULL link of a node is replaced with the address of its inorder predecessor.
- A right NULL link of a node is replaced with the address of its inorder successor.

Left Link (Predecessor)	Data	Right Link (Successor)



Threaded Binary Trees

- **Node Structure of Threaded Binary Tree :**

One way of representing the node structure will be :

```
structure tree
{
        int info ;
        int left-thread,right-thread;
        struct tree *left, *right;
};
```

Threaded Binary Trees

- Advantages:

1. Non-recursive preorder traversal can be implemented with out a stack.
2. Non-recursive inorder traversal can be implemented with out a stack.
3. Non-recursive postorder traversal can be implemented with out a stack.

- Disadvantages:

1.Insertion and Deletion operation becomes more difficult.

2. Tree traversal algorithms be difficult.

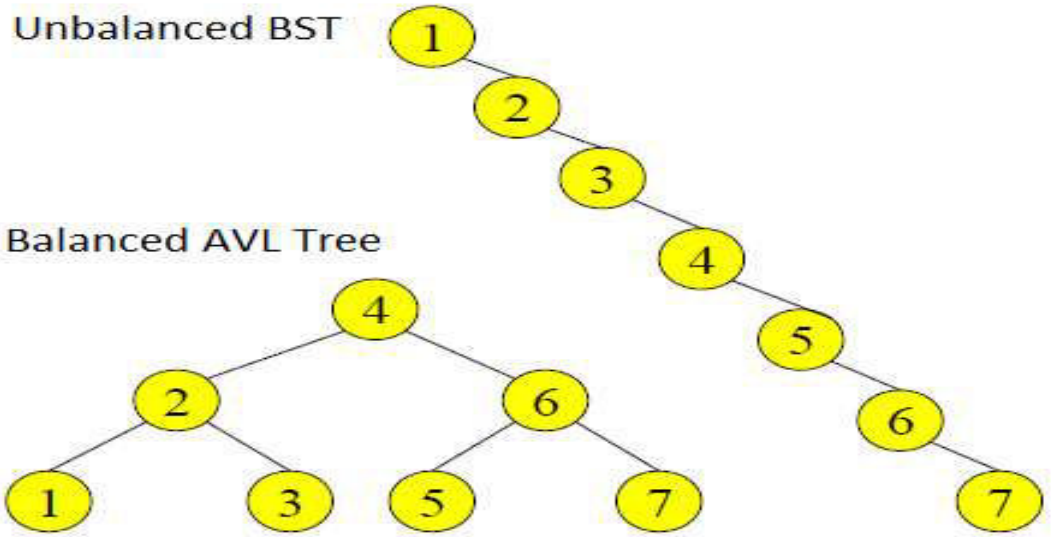
3.Memory required to store a node increases. Each node has to store more information, whether they are normal links or threaded links.

Binary Search Tree-Best Time

- All BST operations are O(d), where d is tree depth
- minimum d is $d = \lfloor \log_2 N \rfloor$ for a binary tree with N nodes
  - What is the best case tree?
  - What is the worst case tree?
- So, best case running time of BST operations is O(log N)

Binary Search Tree-Worst case Time

- Worst case running time is O(N)
- What happens when you insert elements in ascending order
- **Eg: insert 1,2,3,4,5,6,7 into an empty BST**
- Problem: Lack of “Balance”



Unit V: Trees

- **Introduction to trees:**
  - Basic Tree Concepts
- **Binary Trees:**
  - Concept &Terminologies
  - Representation of Binary Tree in memory
  - Traversing a binary tree
- **Binary Search Trees (BST):**
  - Basic Concepts
  - BST operations
  - Concept of Threaded Binary Search Tree
- **AVL Tree:**
  - Basic concepts and rotations of a Tree

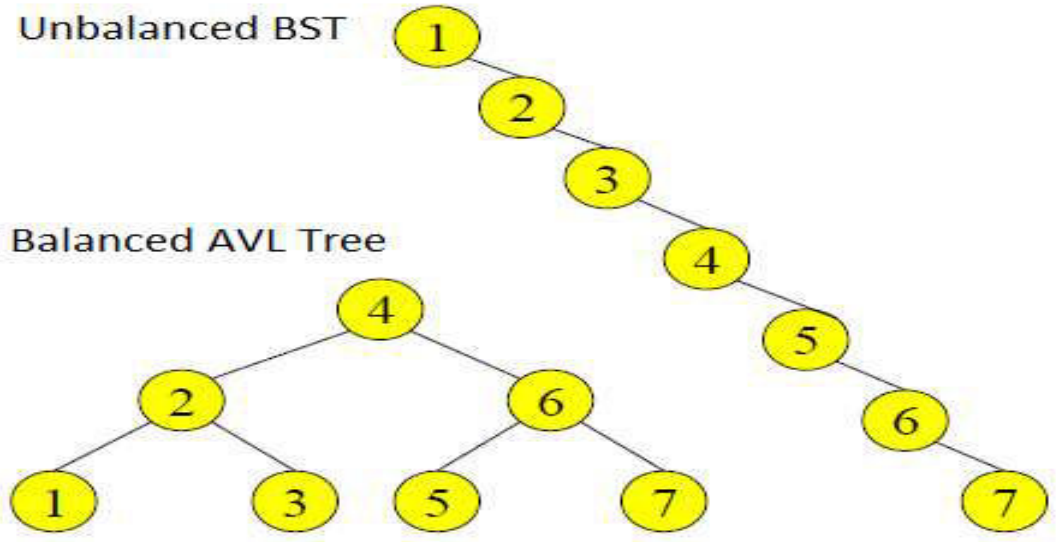
AVL Trees

- Named after 2 Russian mathematicians in 1962
- Georgii Adelson Velsky & E.M.Landis (AVL)
- AVL trees are height-balanced Binary Search Tree

Balance Factor(BF)

=height of left subtree - height of right subtree

balance factor = h_{left}-h_{right}



AVL Trees…

After performing any operation on AVL tree, the balance factor of each node is checked.

There are following two cases possible-

Case-01:

After the operation, the balance factor of each node is either 0 or 1 or -1.

- In this case, the AVL tree is considered to be balanced.
- The operation is concluded.

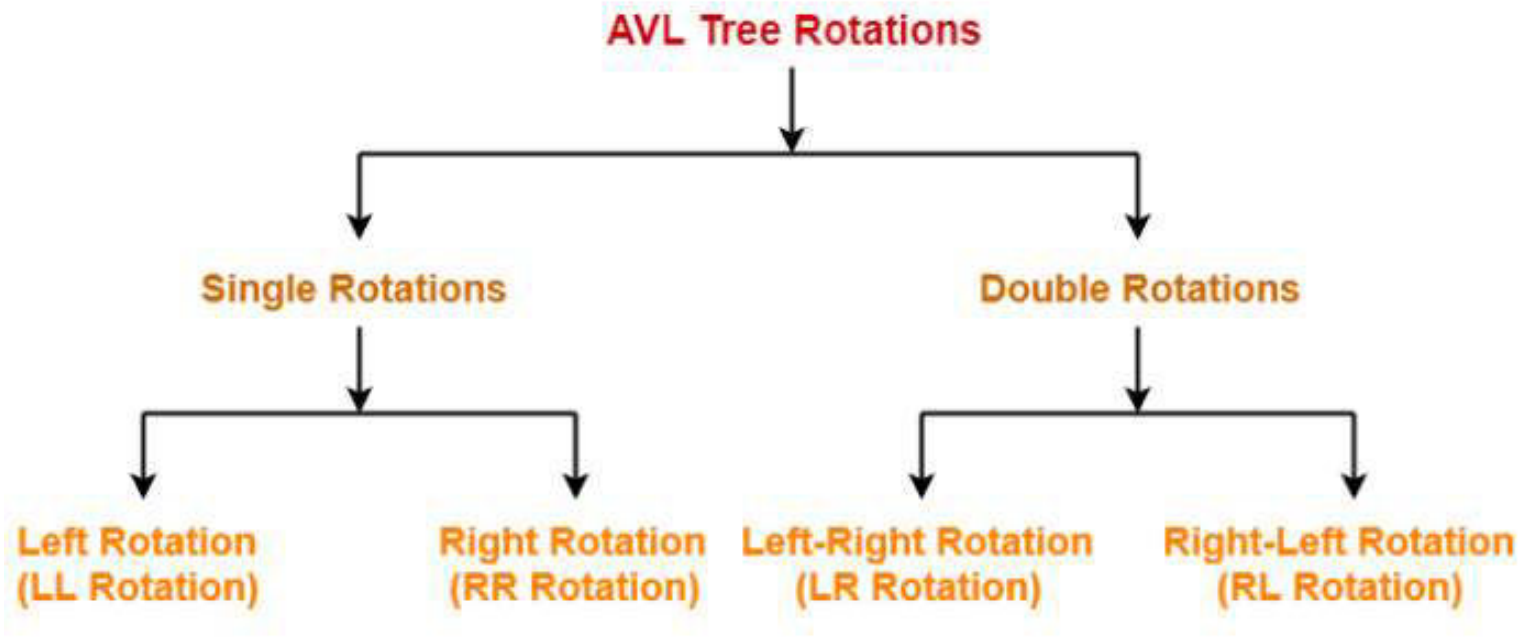
Case-02:

After the operation, the balance factor of at least one node is not 0 or 1 or -1.

- In this case, the AVL tree is considered to be imbalanced.
- Rotations are then performed to balance the tree.

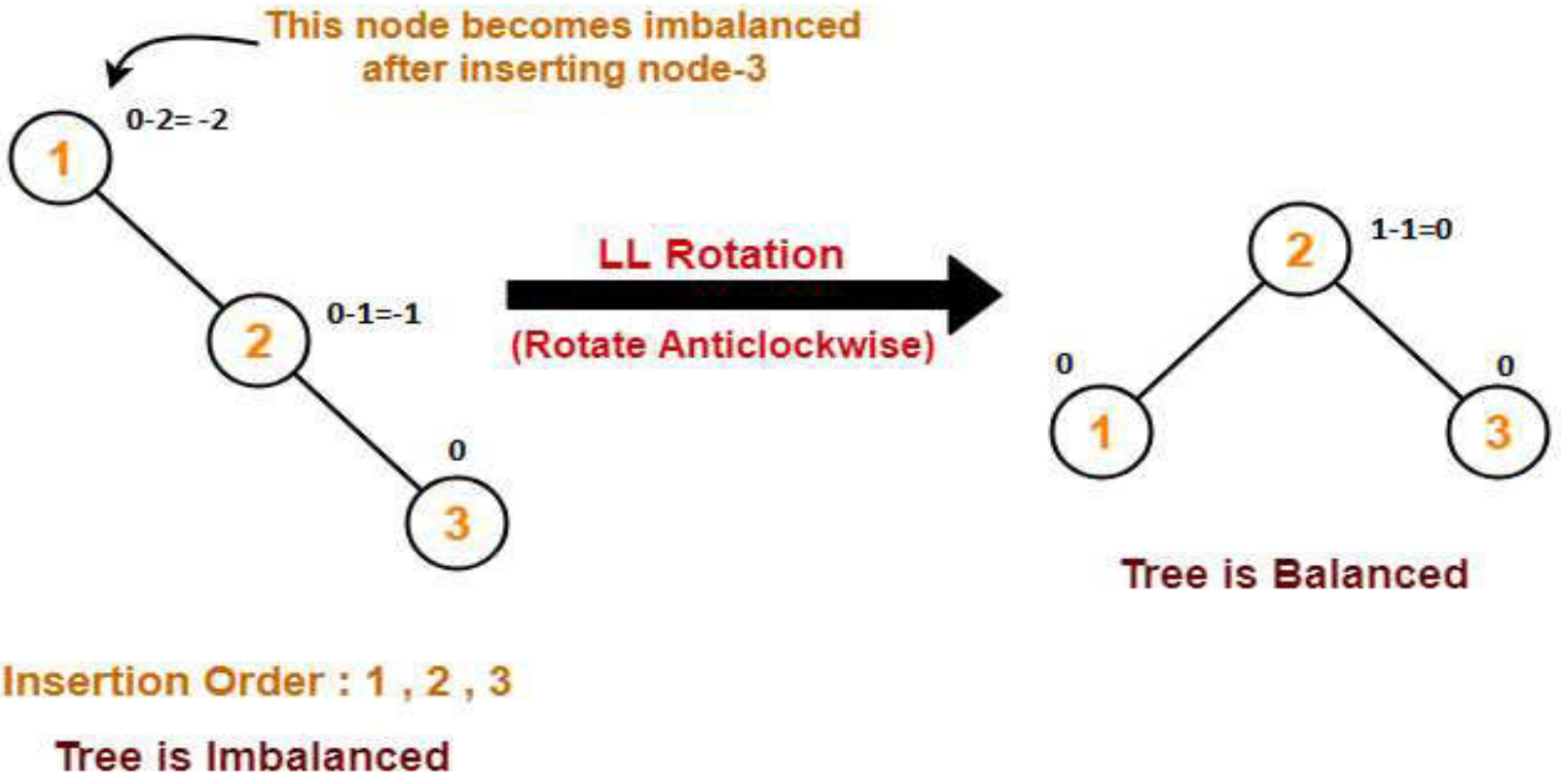
AVL Trees Tree Rotation

- Rotation is the process of moving the nodes to make tree balanced.
- There are 4 kinds of rotations possible in AVL Trees:
  - Left Rotation (LL Rotation)
  - Right Rotation (RR Rotation)
  - Left-Right Rotation (LR Rotation)
  - Right-Left Rotation (RL Rotation)



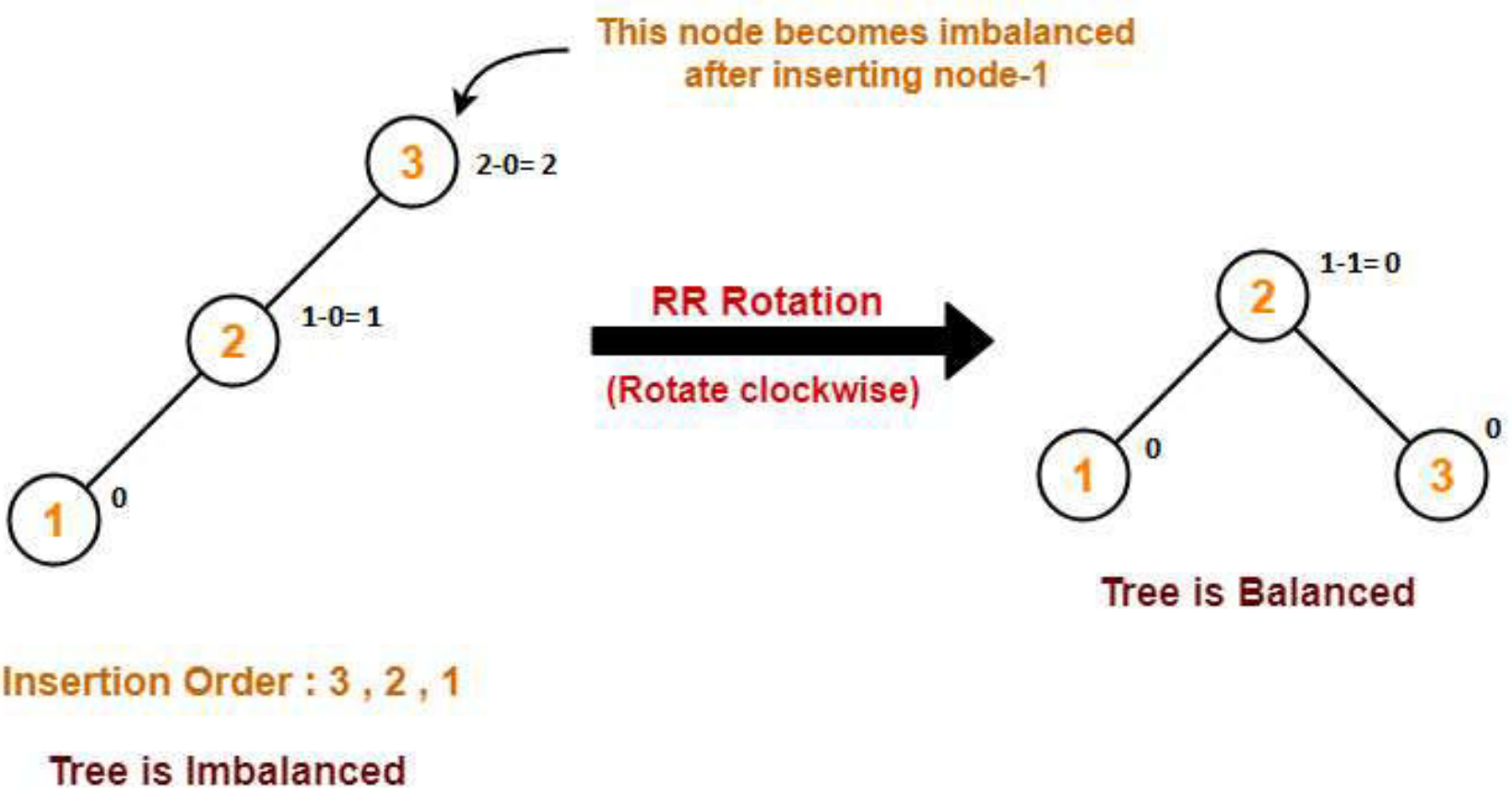
AVL Tree Rotation

- **Cases Of Imbalance and Their Balancing Using Rotation Operations-**
- **Case-01: Insert 1,2,3**



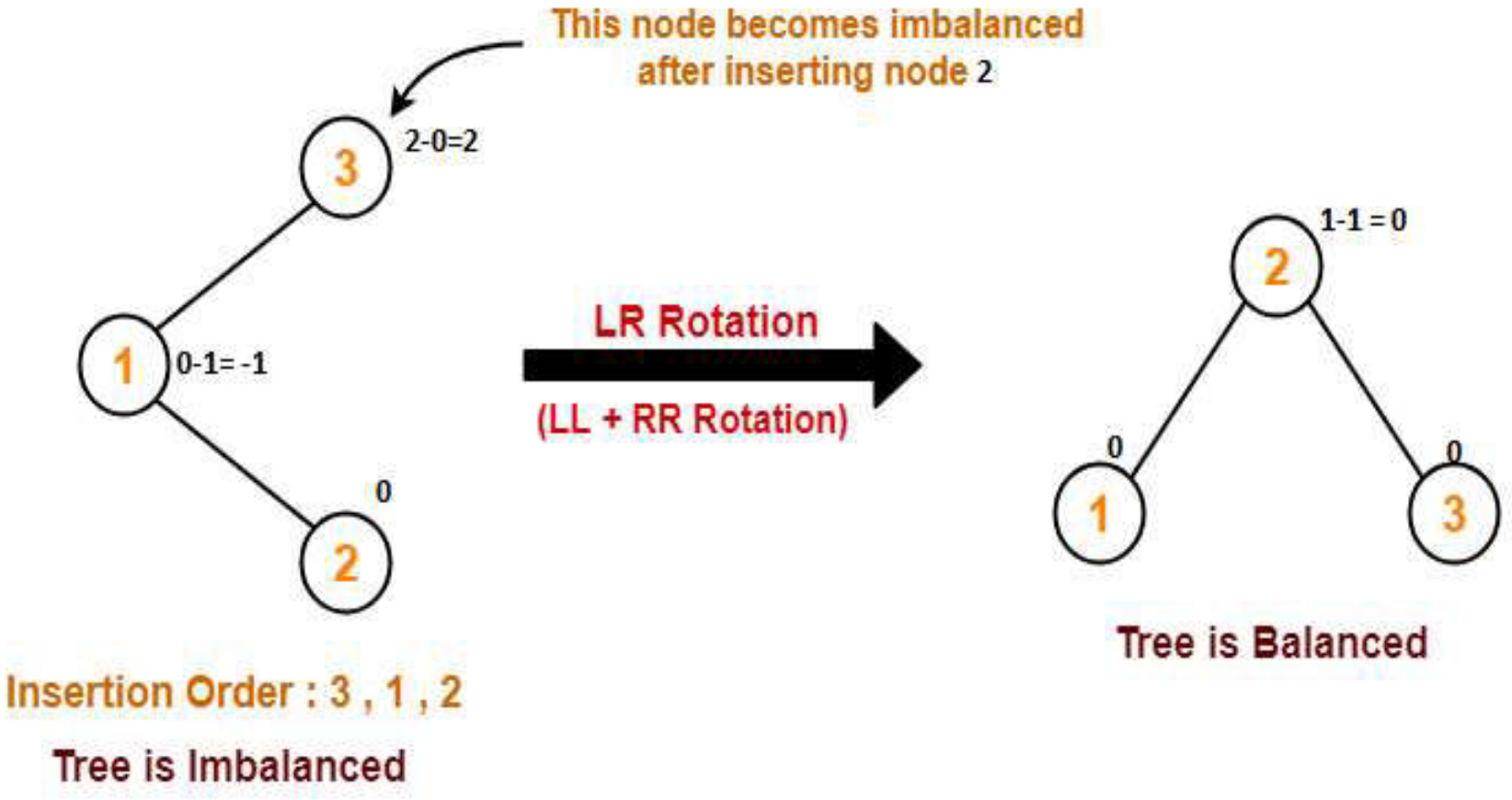
AVL Tree Rotation…

- **Case-02: Insert 3,2,1**



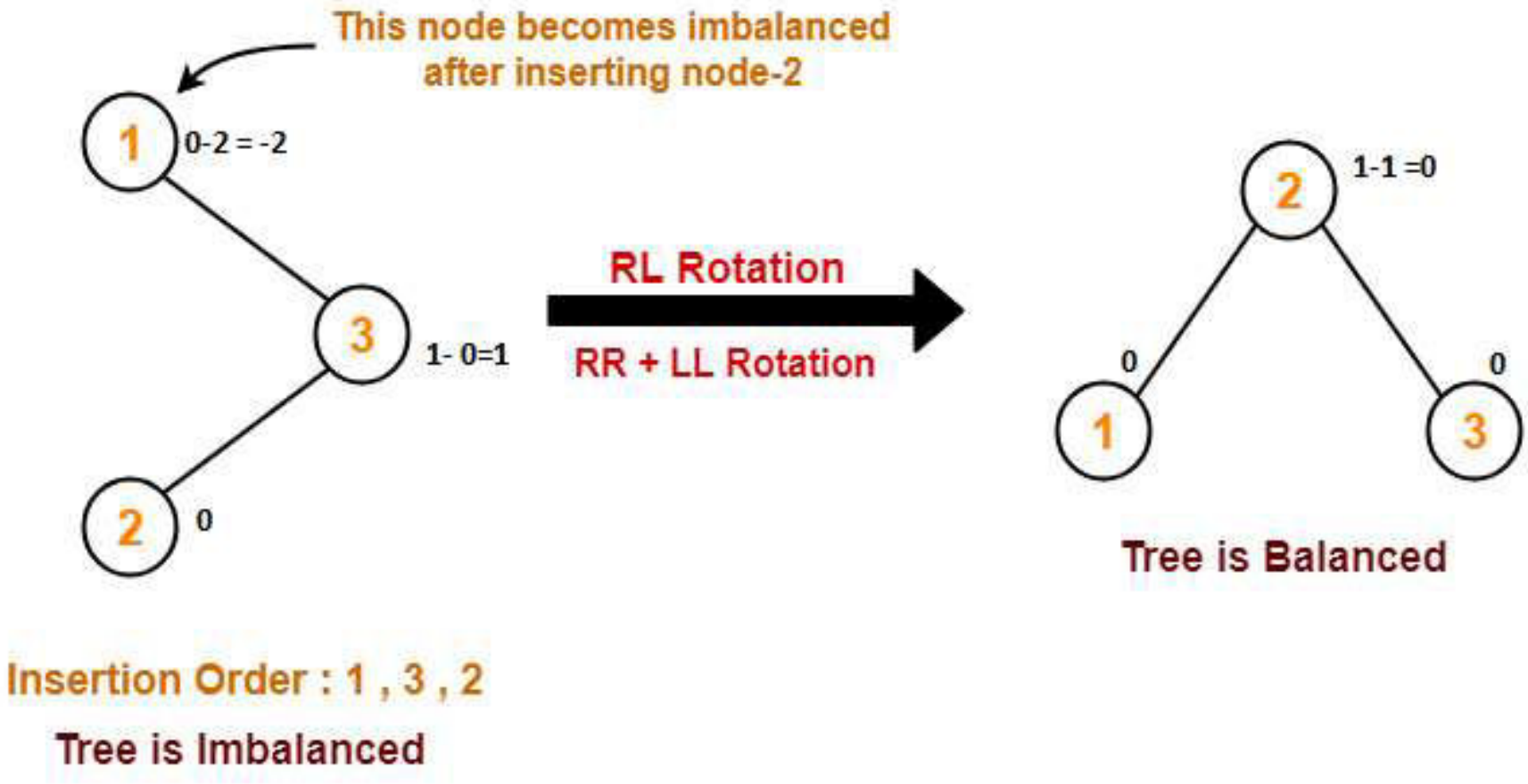
AVL Tree Rotation…

- **Case-03: Insert 3,1,2**



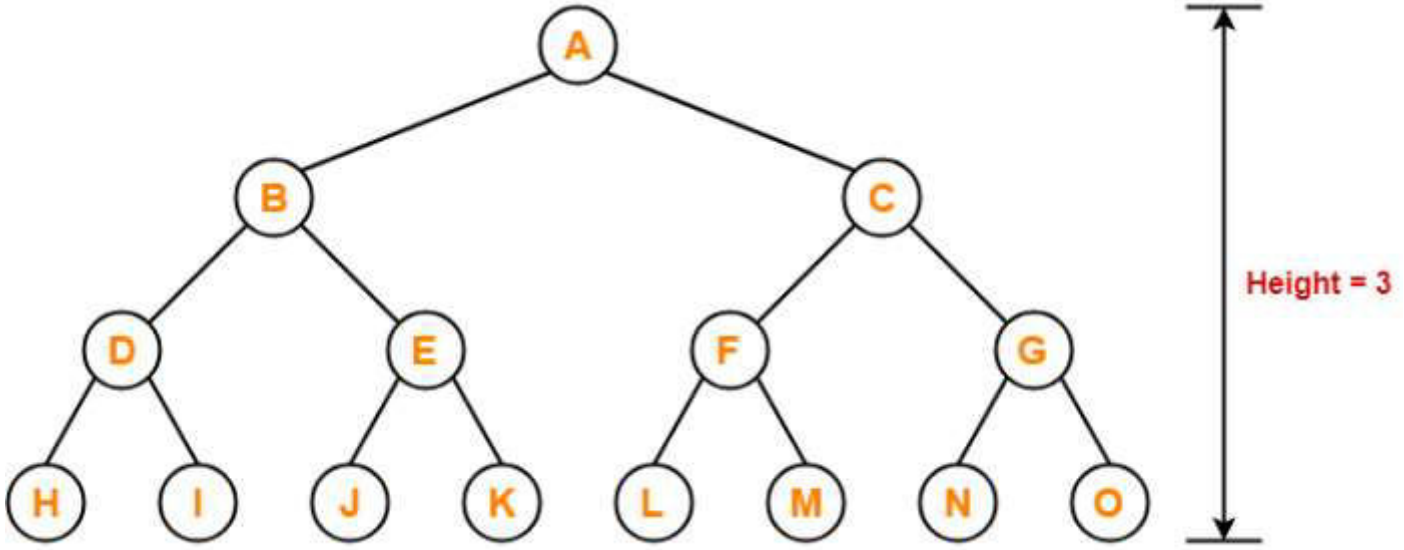
AVL Tree Rotation…

- **Case-04: Insert 1,3,2**



AVL Tree Properties

- **Property-01**
- **Maximum possible number of nodes in AVL tree of height H = $2^{H+1} - 1$**
- **Example-**
- Maximum possible number of nodes in AVL tree of height =3
  $= 2^{3+1} - 1$
  $= 16 - 1$
  $= 15$
- Thus, **in AVL tree of height =3, maximum number of nodes that can be inserted = 15.**
- **We can not insert more number of nodes in this AVL tree**.

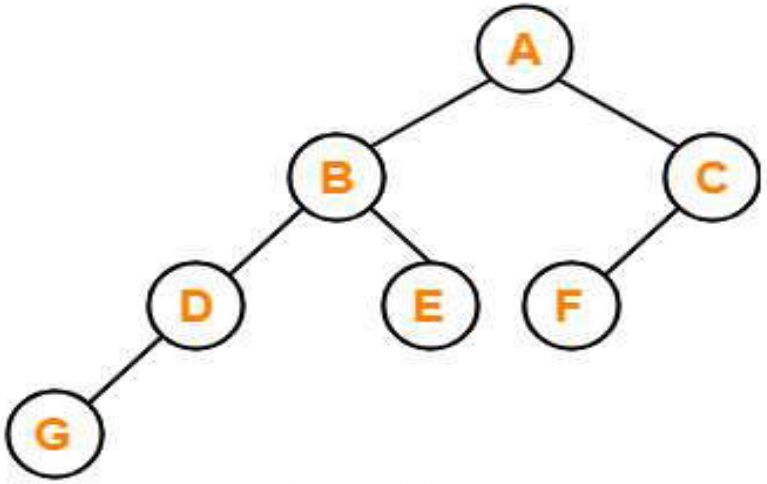


AVL Tree Properties

Property-02:

- **Minimum number of nodes in AVL Tree of height H is given by a recursive relation-**
- **N(H) = N(H-1) + N(H-2) + 1**

- Base conditions for this recursive relation are: N(0) = 1 & N(1) = 2
- **Example-**
- Minimum possible number of nodes in AVL tree of height 3 = 7



AVL Tree
(Height = 3)

AVL Tree Properties

- **Property-03:**
- **Minimum possible height of AVL Tree using N nodes = $\lfloor \log_2 N \rfloor$**

- **Example-**
- Minimum possible height of AVL Tree using 8 nodes

$= \lfloor \log_2 8 \rfloor$
$= \lfloor \log_2 2^3 \rfloor$
$= \lfloor 3 \log_2 2 \rfloor$
$= \lfloor 3 \rfloor$
$= 3$

AVL Tree Properties

- **Property-04:**
- **Maximum height of AVL Tree using N nodes is calculated using recursive relation**

  N(H) = N(H-1) + N(H-2) + 1

- Base conditions for this recursive relation are:
- N(0) = 1
- N(1) = 2

- **NOTE:**
- If there are n nodes in AVL Tree, its maximum height can not exceed $1.44\log_2 n$.
- In other words, Worst case height of AVL Tree with n nodes = $1.44\log_2 n$.

Insertion in AVL Tree

- Insertion Operation is performed to insert an element in the AVL Tree.
- To insert an element in the AVL tree, follow the following steps:
  - Insert the element in the AVL tree in the same way the insertion is performed in BST.
  - After insertion, check the balance factor of each node of the resulting tree.

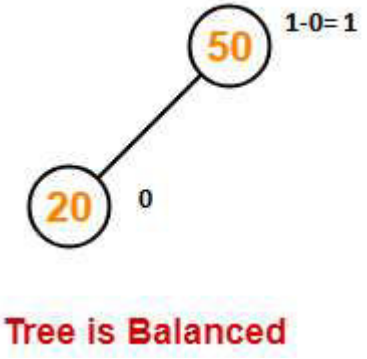
Insertion in AVL Tree

- Construct AVL Tree for the following sequence of numbers: 50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48
- **Solution-**

Step-01: Insert 50

Step-02: Insert 20

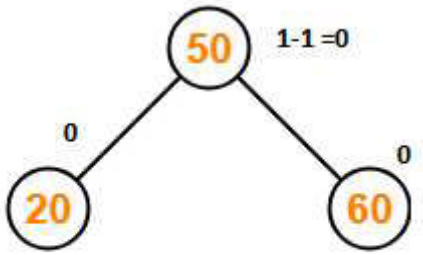
As 20 < 50, so insert 20 in 50's left sub tree.



Insertion in AVL Tree

Step-03: Insert 60

As 60 > 50, so insert 60 in 50's right sub tree.

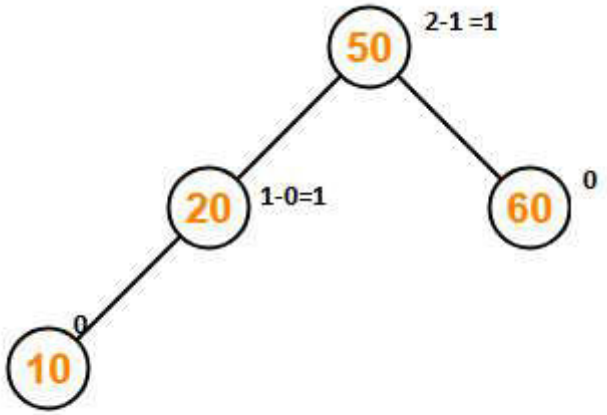


Tree is Balanced

Step-04: Insert 10

As 10 < 50, so insert 10 in 50's left sub tree.

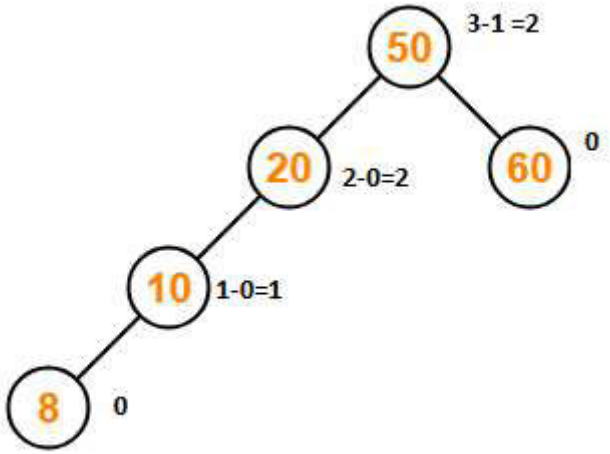
As 10 < 20, so insert 10 in 20's left sub tree.



Tree is Balanced

Insertion in AVL Tree

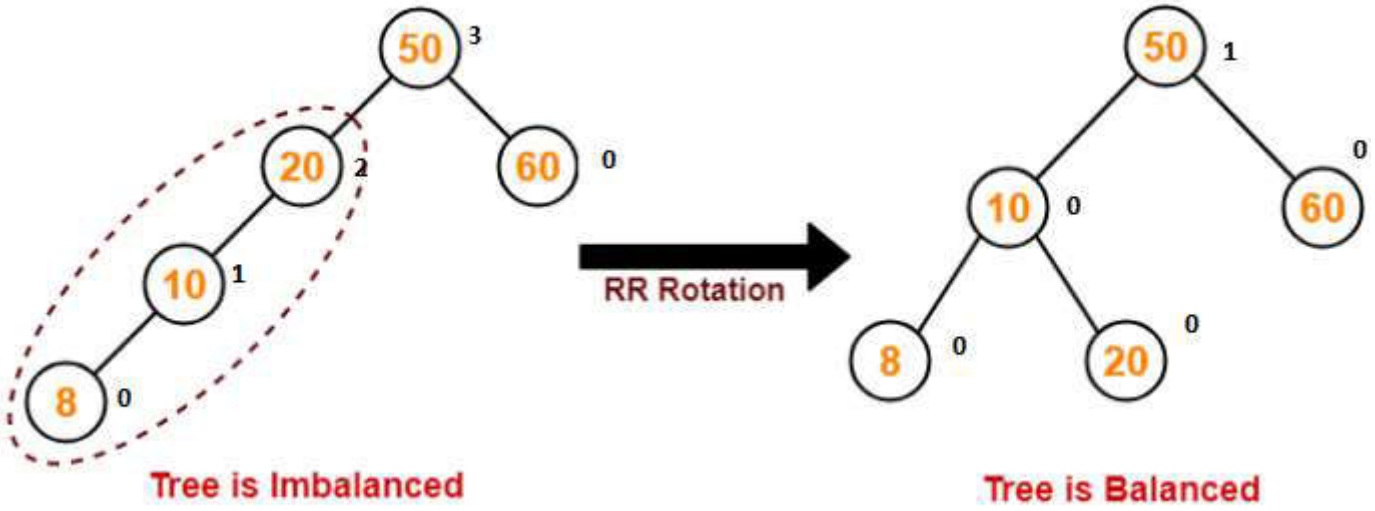
- **Step-05: Insert 8**
- As 8 < 50, so insert 8 in 50's left sub tree.
- As 8 < 20, so insert 8 in 20's left sub tree.
- As 8 < 10, so insert 8 in 10's left sub tree.



Tree is Imbalanced

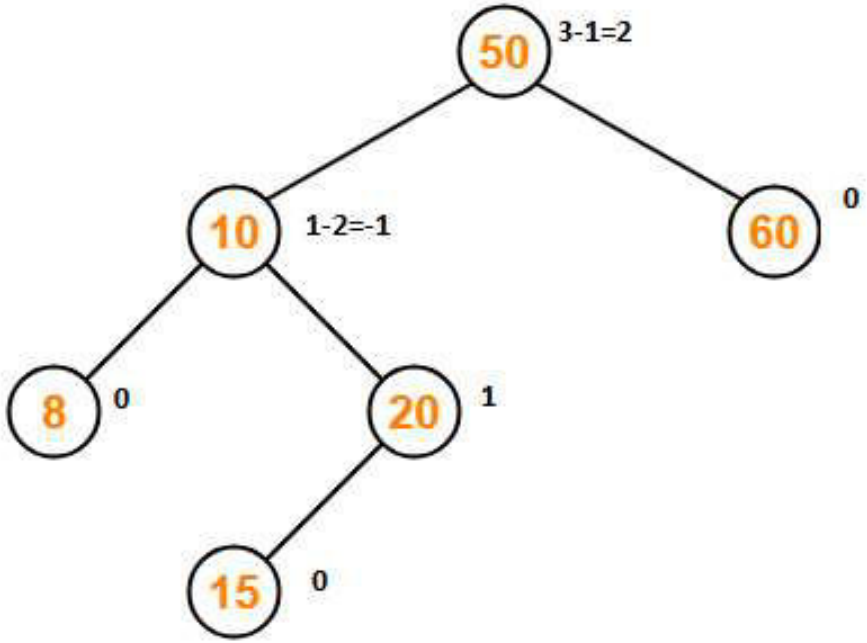
Insertion in AVL Tree

- To balance the tree,
- Find the first imbalanced node on the path from the newly inserted node (node 8) to the root node.
- The first imbalanced node is node 20.
- Now, count three nodes from node 20 in the direction of leaf node.
- Then, use AVL tree rotation to balance the tree.



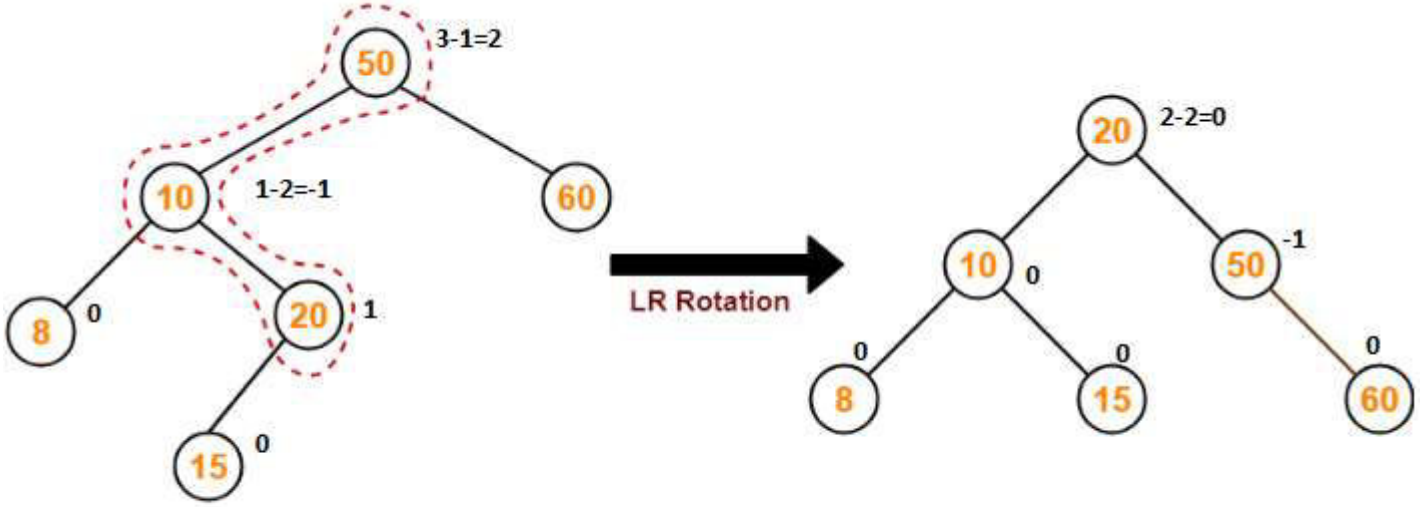
Insertion in AVL Tree

- **Step-06: Insert 15**
- As 15 < 50, so insert 15 in 50's left sub tree.
- As 15 > 10, so insert 15 in 10's right sub tree.
- As 15 < 20, so insert 15 in 20's left sub tree



Insertion in AVL Tree

- To balance the tree,
- Find the first imbalanced node on the path from the newly inserted node (node 15) to the root node.
- The first imbalanced node is node 50.
- Now, count three nodes from node 50 in the direction of leaf node.
- Then, use AVL tree rotation to balance the tree.

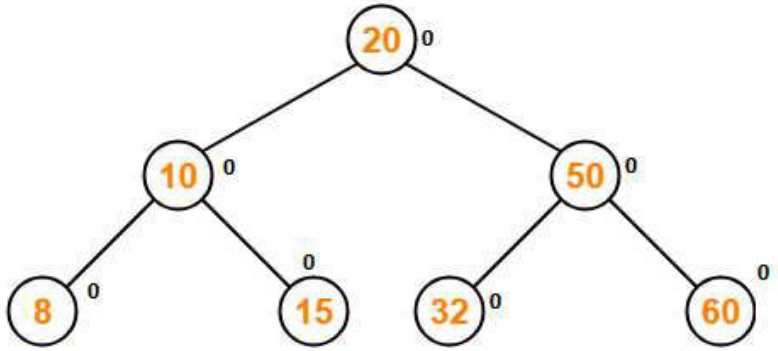


Insertion in AVL Tree

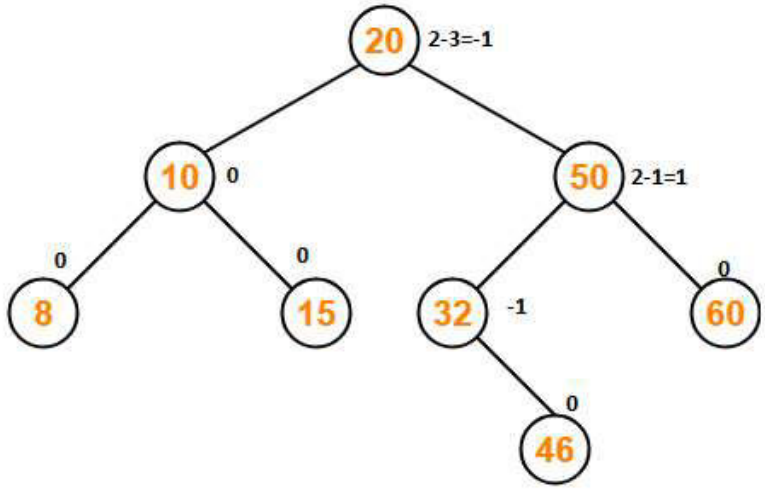
- **Step-07: Insert 32**
- As 32 > 20, so insert 32 in 20's right sub tree.
- As 32 < 50, so insert 32 in 50's left sub tree.

Step-08: Insert 46

- As 46 > 20, so insert 46 in 20's right sub tree.
- As 46 < 50, so insert 46 in 50's left sub tree.
- As 46 > 32, so insert 46 in 32's right s



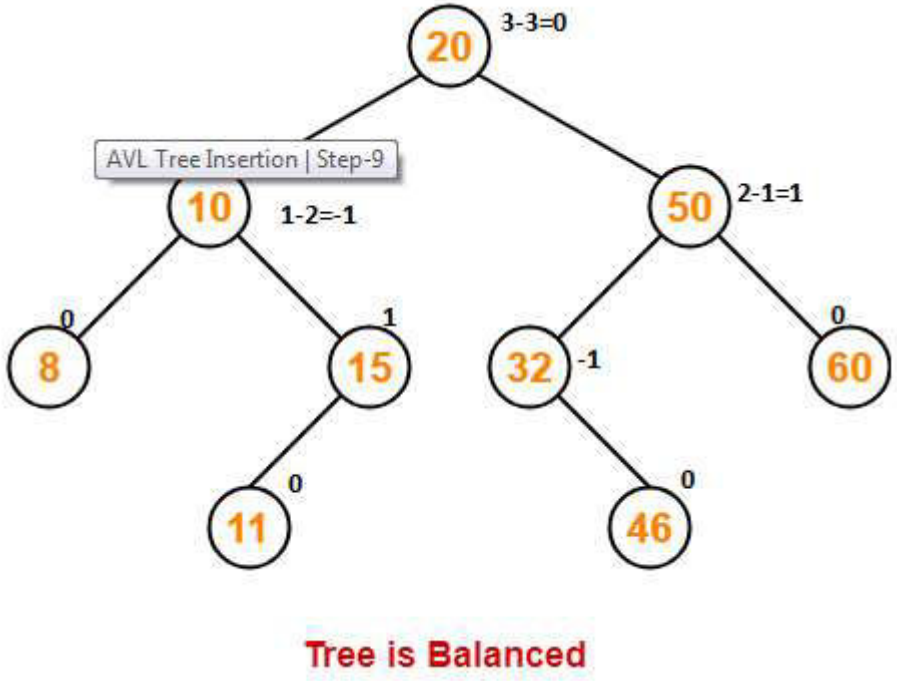
Tree is Balanced



Tree is Balanced

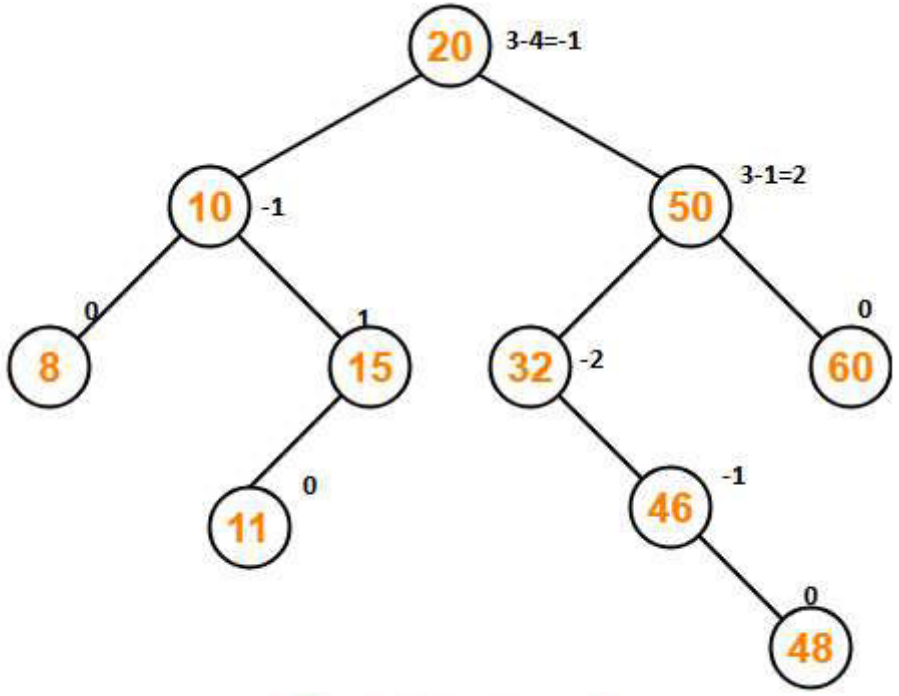
Insertion in AVL Tree

- **Step-09: Insert 11**
- As 11 < 20, so insert 11 in 20's left sub tree.
- As 11 > 10, so insert 11 in 10's right sub tree.
- As 11 < 15, so insert 11 in 15's left sub tree.



Insertion in AVL Tree

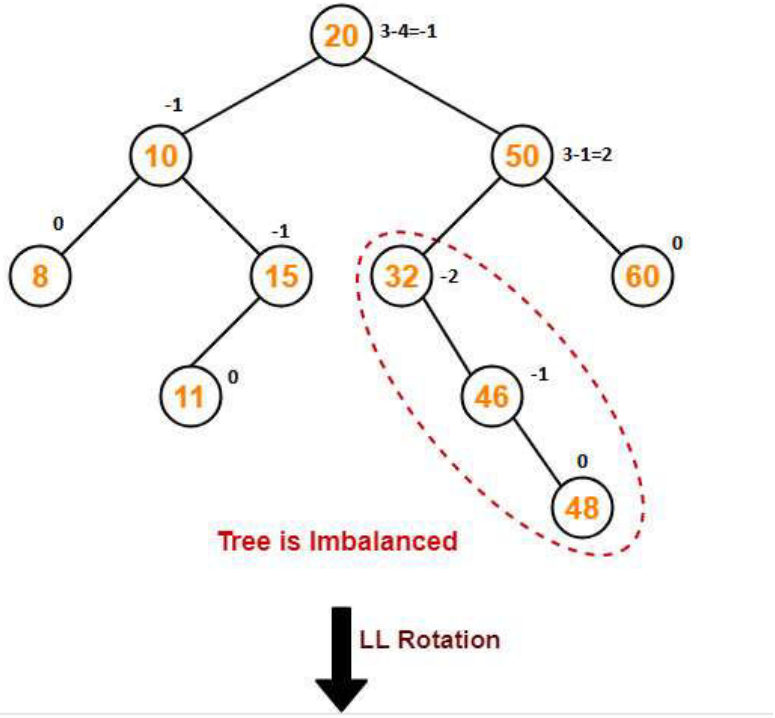
- **Step-10: Insert 48**
- As 48 > 20, so insert 48 in 20's right sub tree.
- As 48 < 50, so insert 48 in 50's left sub tree.
- As 48 > 32, so insert 48 in 32's right sub tree.
- As 48 > 46, so insert 48 in 46's right sub tree.



Tree is imbalanced

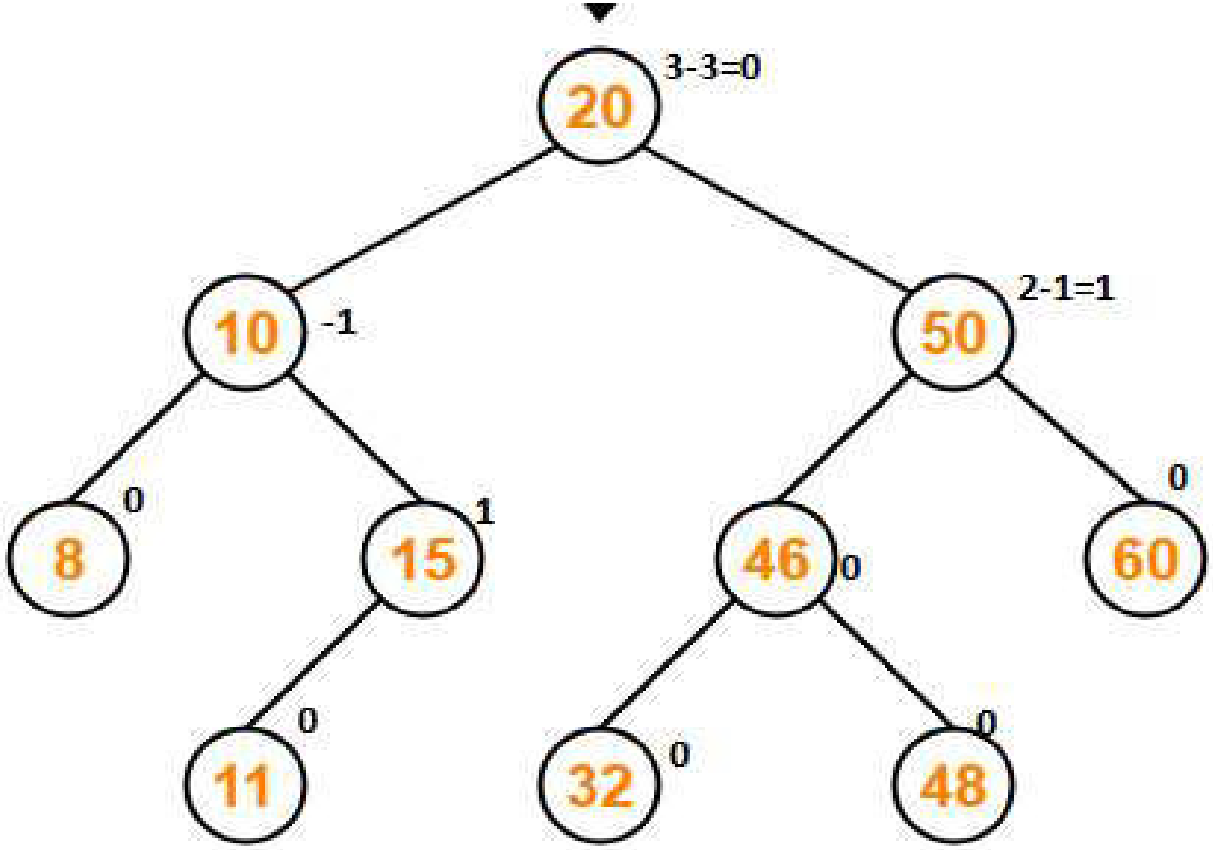
Insertion in AVL Tree

- To balance the tree,
- Find the first imbalanced node on the path from the newly inserted node (node 48) to the root node.
- The first imbalanced node is node 32.
- Now, count three nodes from node 32 in the direction of leaf node.
- Then, use AVL tree rotation to balance the tree.



Insertion in AVL Tree

- This is the final balanced AVL tree after inserting all the given elements.



Tree is Balanced

Pros and Cons of AVL Tree

Arguments for AVL trees:

1. Search is O(log N) since AVL trees are always balanced.
2. Insertion and deletions are also O(logn)
3. The height balancing adds no more than a constant factor to the speed of insertion.

Arguments against using AVL trees:

1. Difficult to program & debug; more space for balance factor.
2. Asymptotically faster but rebalancing costs time.
3. Most large searches are done in database systems on disk and use other structures (e.g. B-trees).
4. May be OK to have O(N) for a single operation if total run time for many consecutive operations is fast (e.g. Splay trees).

Questions Mapping with Bloom's Taxonomy Level

InSem (30marks) and EndSem Exam (70) marks)

1. **Define BST (BT1)**
2. **Explain with an example** how nodes **are inserted AVL Tree (BT2)**
3. **Construct** the **BST** from the following set of strings: JAN,FEB,MARCH,APRL **(BT3)**
4. **Construct BT** if the following are given: **(BT3)**
    1. **Inorder(LDR):40,20,80,50,30,60,90,30,70**
    2. **Preorder(DLR):10,20,40,50,80,30,60,90,70**
5. Write a non recursive algorithm for : (BT4)

   (i)Inorder traversal (ii) Post order traversal (iii)Preorder traversal
6. Write a 'C' pseudo-algorithm to insert a node in inorder threaded binary search tree. (BT5)
7. Write a program to count the no. of terminal nodes in a binary tree T. (BT6)

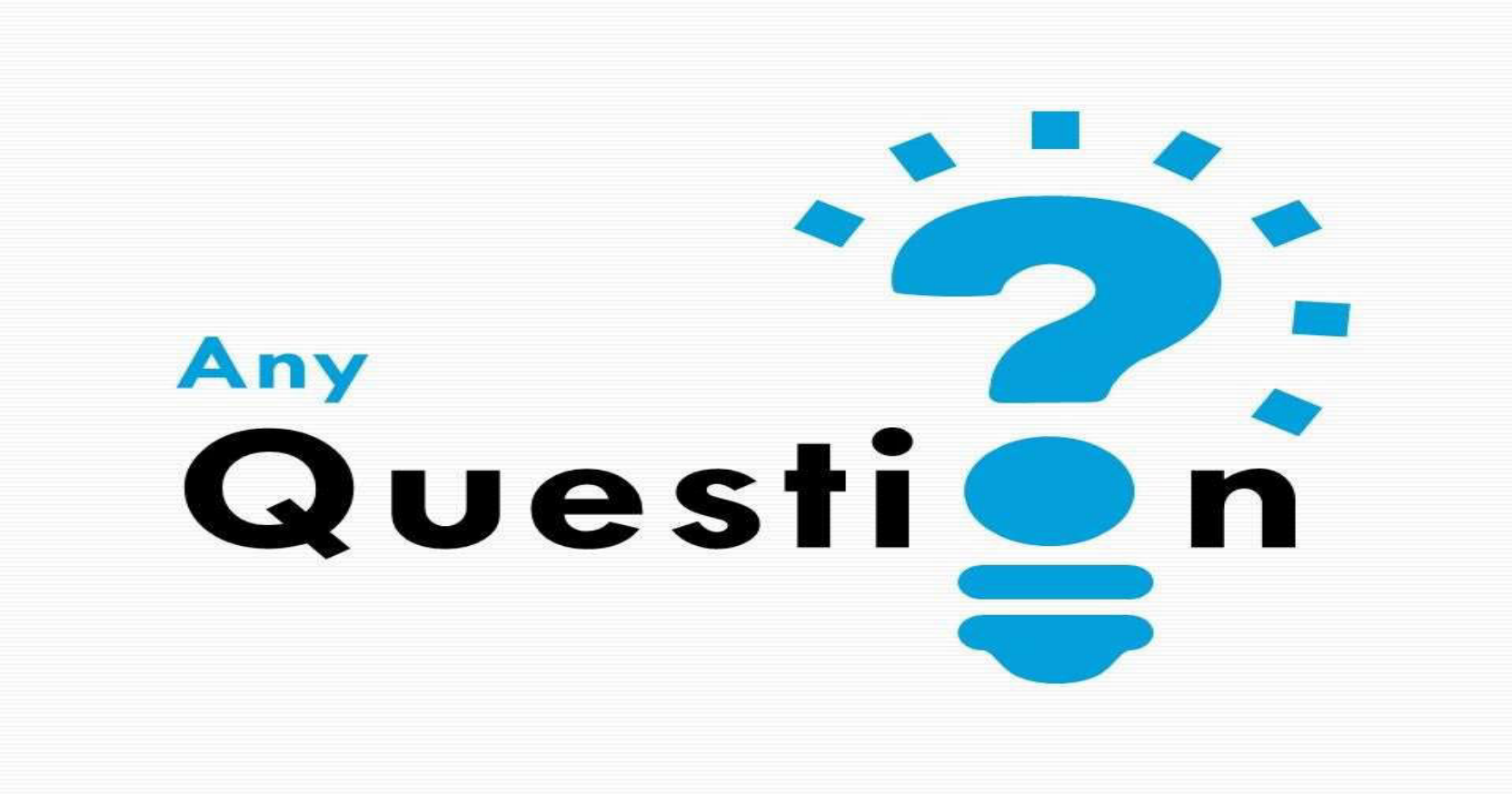
Effective Teaching Techniques

1. **Use of animation**
   - Yashavant Kanetkar, "Data Structures Through C", BPB Publication, 2nd Edition(CD available)
   - Wacom (Pen Tablet)
2. **At the end of Topic/Unit**
   - MCQs using PPT
   - Online Test

     Eg: https://www.indiabix.com/online-test/c-programming-test/
3. **Video Lectures by experts (MOOC / NPTEL/Coursera/Any good university /Guest Lectures)**
   - NPTEL Course **"Programming & Data Structure"**

     https://nptel.ac.in/courses/106/105/106105085/
   - NPTEL Course "**Data Structures & Algorithms"** https://nptel.ac.in/courses/106/102/106102064/
4. **Execution of Programs (eg: codeBlocks)**
5. **VIRTUAL LAB**: (A few experiments related to the course available on Virtual labs):
   - **1. Data Structures-I:** https://ds1-iiith.vlabs.ac.in/data-structures-1/
   - **2.Data Structures -II:** https://ds2-iiith.vlabs.ac.in/data-structures-2/
   - **3. Data Structures Lab:**http://cse01-iiith.vlabs.ac.in/
   - **4. Computer Programming Lab**: http://cse02-iiith.vlabs.ac.in/

Discussion Time



Other operations on binary tree

1) Finding the depth of a tree

```
void_depth_tree(struct node *root, int l)

// root being the pointer to the root of the tree
// l is the level value, initially passed by the calling function as 0
// The method is to check for leaf node and its level, if it is greater than the depth at
// present then assign this level to the depth. Note that depth is a global variable initialised // to
0
        if(root)
        {       if((root->left==NULL)&&(root(right==NULL))
                {
                        // it is a leaf node
                        if (level > depth)
                                depth=level;
                }
                        else
                        {       depth_tree(root->left,level+1);
                                depth_tree(root->right,level+1);
                        }
        }
}
```

Other operations on binary tree

2) Finding the mirror image of the tree

```
void mirror(struct node *root)
// root is the pointer to the tree
{
        if(root)
        {
                mirror(root->left);
                mirror(root->right);
                // q is pointer of the type node
                q=root->left;
                root->left=root->right;
                root->right=q;
        }
}
```

Other operations on binary tree

3) Level wise printing of the nodes :

```
void disp(struct node *root)
{
        add(root); // function to add a node to a queue
        while(queue is not empty)
        {
        x=delete();// function to delete a node from the queue
                printf("%d",x);
                if(root->left!=NULL)
                        add(root->left);
                if(root->right!=NULL)
                        add(root->right);
        }
}
```

Other operations on binary tree

4) Copying a tree.

```
public BinaryTree copyTree(BinaryTree bt)
{
if (bt == null) return null;
BinaryTree left = copyTree(bt.leftChild);
BinaryTree right=copyTree(bt.rightChild);
return new BinaryTree(bt.value, left, right);
}
```

Other operations on binary tree

5) Checking equality :

The pseudo code for this may be written as :

```
int equal (struct node *q1, struct node *q2)
{       // q1 & q2 point to the two subtrees.
        If ((q1) && (q2))
        {       if (q1 → data = q2 → data)
                {
                        equal (q1 → left, q2 → left);
                        equal (q1 → right, q1 → right);
                }
                else
                        return (– 1);  //not equal
        }
        return (1);
}
```

Unit V: Trees

- **Inorder Traversal of a Threaded Binary Tree :**

```
void inorder (int P)
{
   if (P > 0) find the / * first leftmost node for inorder traversal. */
                        while (ws [P].left >0)
                        P = ws[P].left;
while (P) /* visit the node go to its successor */.
            {
                                                visit (P);
                                                P = ws[P].right;
            if (P > 0) /* if it is a thread link it gives the successor */
                                    P = – P;
            else
            if(P > 0) /* else move as far left as possible */
                                    while (ws [P].left > 0)
                                    P = ws[P].left;
             /* If neither section is done, P = 0 then traversal is done */
            }
}
```

Other operations on binary tree

Preorder Traversal of a Threaded Binary Tree

```
void preorder (int P)
{
        while (P > 0)
        {
                visit (P);
                if (ws [P].left > 0)
                P = ws [P].left;
        else
                if (ws [P].right > 0)
                P = [P].right;
/ *     otherwise, P is a leaf. We take its right thread, which will return to a node already
visited and move to right again */
        else
        {
                while (ws [P].right < 0)
                P = – ws [P].right;
                P = ws [P].right;
        }
}
}
```

Other operations on binary tree

- **Postorder Traversal of Binary Tree with Inorder Threads**

```
void post (int p)
{
next action = GOLEFT ; /* an enumerated data type */
while(p)
switch (next action)
{
        case GOLEFT : /* Traverse left subtree */
        If (ws [p].left > 0)
        p = ws [p].left;
        else
        next action = GORIGHT;
        break;
        Case GORIGHT : /* Right subtree traversal */
        If (ws [p].right > 0)
        {
                p = ws [p].right;
                next action = GOLEFT;
        }
        else
        next action = VISIT;
        break;
        case VISIT:
        visit (p);
        parent (p, sp, snextaction);
        break;
        }
}
```

Stay Safe &Healthy, Happy learning & Enjoy Teaching!!