Unit –IV

TREES

Objective:

- To impart knowledge of linear and non-linear data structures.

Syllabus:

Binary Trees: Basic tree concepts, properties, representation of binary trees using arrays and linked list, binary tree traversals.

Binary Search Trees: Basic concepts, BST operations: search, insertion, deletion and traversals, creation of binary search tree from in-order and pre (post) order traversals.

Learning Outcomes:

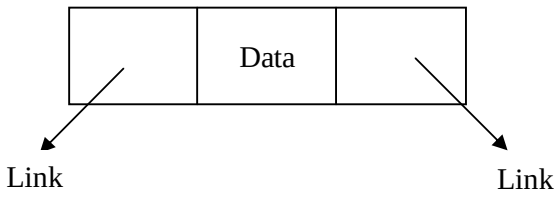
At the end of the unit student will be able to:

1. represent Binary Tees using Arrays and Linked Lists.
2. implement operations on Binary Search Trees.
3. construct Binary Search Trees from its Traversals

Learning Material

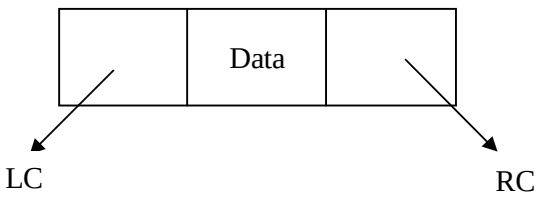
Basic Terminology:

1. Node: It is a main component of tree. It stores the actual data and links to other nodes.



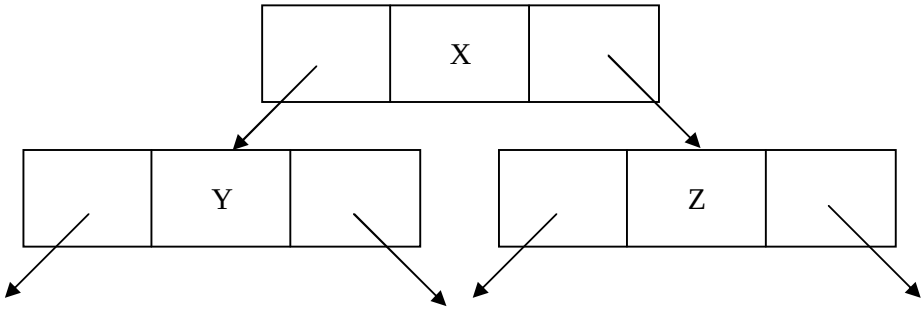
Structure of a node in Tree

2. Link / Edge / Branch: Link is point to other nodes in a tree.



Here *LC* Points To **Left Child** and *RC* Points To **Right Child**.

3. Parent Node: The Immediate Predecessor of a Node is called as Parent Node.

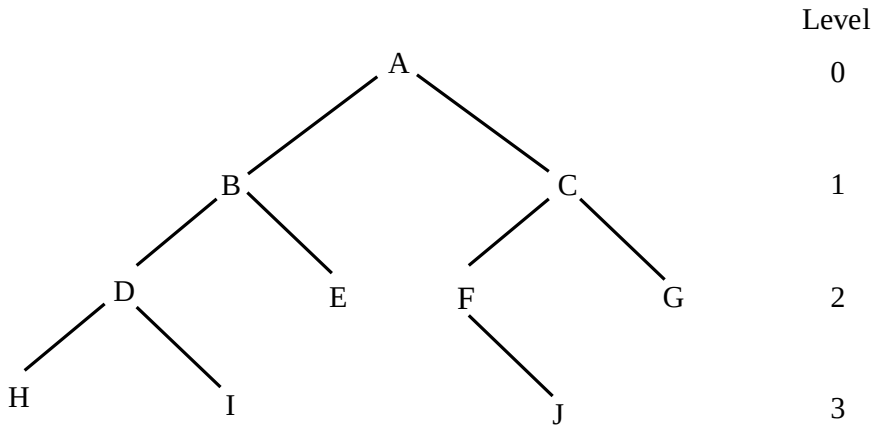


Here X is Parent Node to Node Y and Z.

4. Child Node: The Immediate Successor of a Node is called as Child Node.

In the above diagram Node Y and Z are child nodes to node X.

5. Root Node: Which is a specially designated node, and does not have any parent node.



In the above diagram node A is a Root Node.

6. Leaf node: The node which does not have any child nodes is called leaf node.

- In the above diagram node H, I, E, J, G are Leaf nodes.

7. Level: It is the rank of the hierarchy and the Root node is present at level **0**. If a node is present at level ***l*** then its parent node will be at the level ***l*-1**and child nodes will present at level ***l*+1**.

8. Height / Depth: The number of nodes in the longest path from Root node to the Leaf node is called the height of a tree.

- Height of above tree is 4.
- Height of a tree can be easily obtained as l_{max} + 1. Where l_{max} is the maximum level of a tree.
- In the above example l_{max}= 3. So height = 3 + 1 = 4

9. Siblings: The nodes which have same parent node are called as siblings.

- In the above example nodes B and C are siblings, nodes D and E are siblings , nodes F and G are siblings , nodes H and I are siblings.

10. Degree / Arity: Maximum number of child nodes possible for anode is called as degree of a node.

TREE:

Tree is a nonlinear data structure.

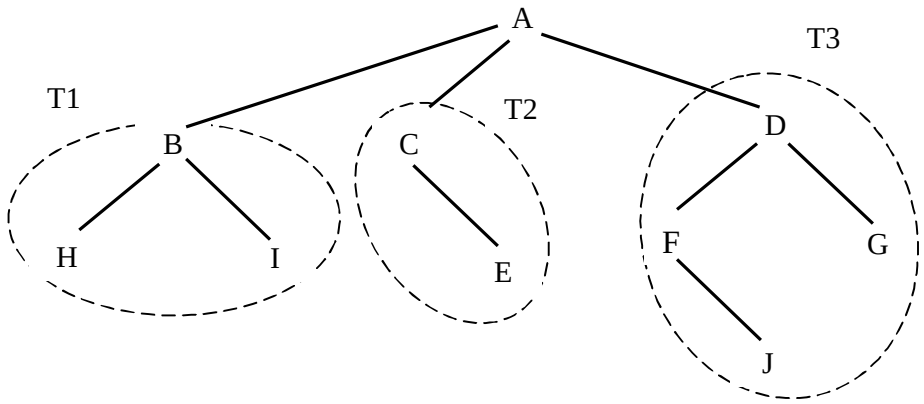
Definition

A tree T is a finite set of one or more nodes such that:

(i) There is a special node called as *root* node.

(ii) The remaining nodes are partitioned into ***n*** disjoint sets T_1, T_2, T_3. . . T_n where n>0.Where each disjoint set is a tree.

T_1, T_2, T_3. . .T_n are called as *sub trees*.



A sample Tree

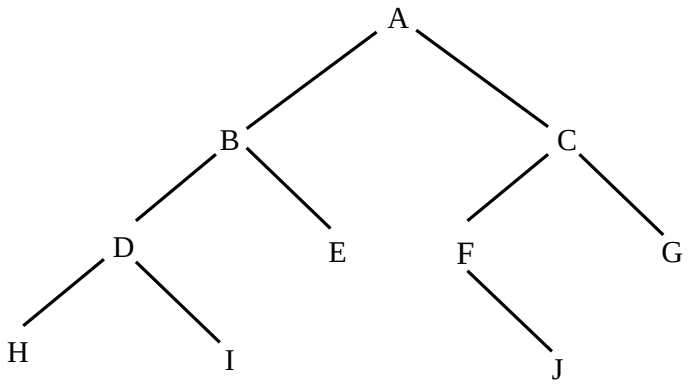
BINARY TREE: Is a special form of a tree.

Definition:

A binary tree is T is a finite set of nodes such that,

(i) T is empty called as empty binary tree.

(ii) T has specially designed node called as Root Node and remaining node of binary tree are partitioned into 2 disjoint sets. One is Left sub tree and another one is Right sub tree.



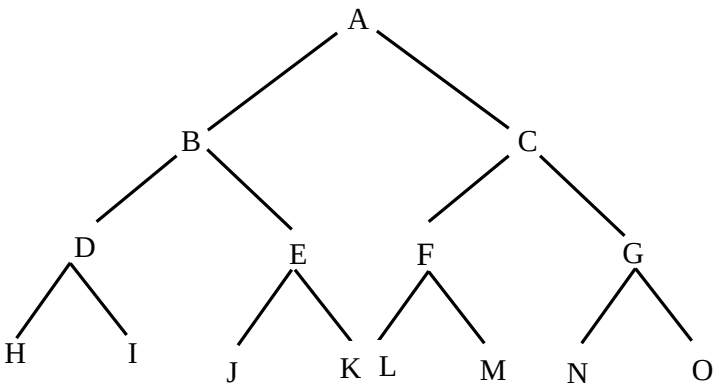
A sample Binary Tree

TWO SPECIAL CASES OF BINAERY TREE

1. Full binary tree
2. Complete binary tree

1. Full Binary Tree: a binary tree is said to be full binary tree, if each level has maximum number of possible nodes.

Eg:

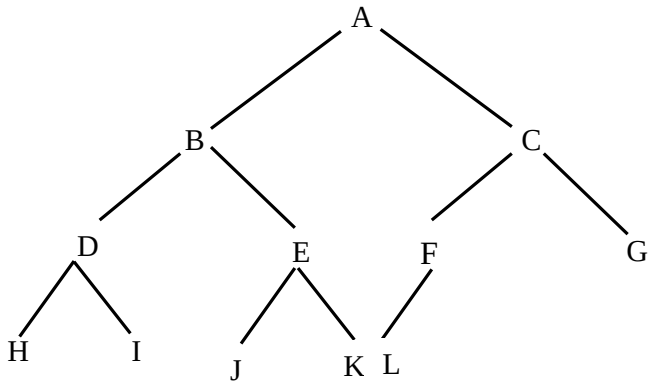


Level	Node's
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$

A Full Binary Tree of height 4

2. Complete Binary Tree: A binary tree is said to be complete binary tree, if all levels except the last level has maximum number of possible nodes, last level nodes are appeared as far left as possible.

Eg:

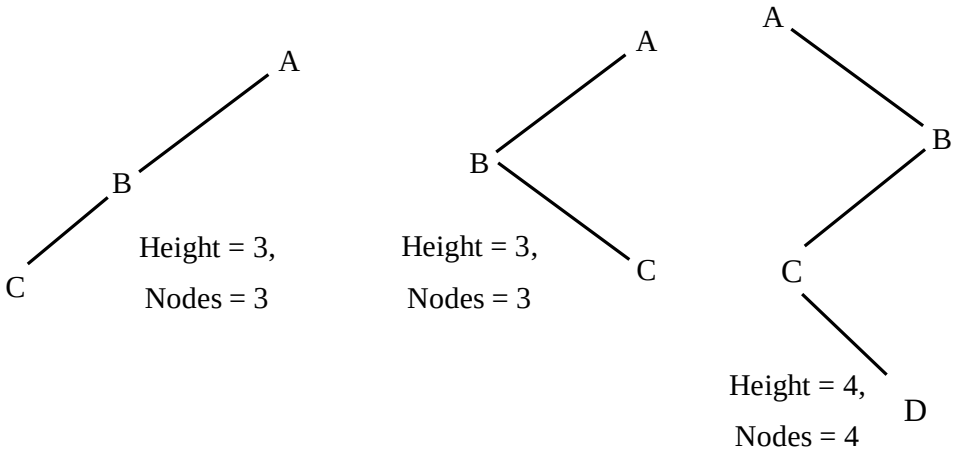


Level	Node's
0	$2^0 = 1$
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$

A Complete Binary Tree of height 4

Properties of Binary Trees

1. In any binary tree, maximum number of nodes on level *l* is 2^l (where *l*>= 0).
2. Maximum number of nodes possible in a binary tree of height *h* is 2^h-1.
3. Minimum number of nodes possible in a binary tree of height *h* is *h*.



.

- Whenever every parent node has only one child, such kind of binary trees are called as ***Skew binary trees***.

4. For any non-empty binary tree, if ***n*** is the number of nodes and ***e*** is the number of edges, then ***n = e + 1***.

   i.e. number of nodes = number of edges +1.
5. For any non-empty binary tree T, if n_0 is the number of leaf nodes (degree = 0) and n_2 is the number of intermediate nodes (degree = 2) then $n_0 = n_2 + 1$.

   i.e. number of leaf nodes = number of non-leaf nodes + 1.
6. The height of a complete binary tree with **n** nodes is $\lceil \log_2(n+1) \rceil$
7. Total number of binary trees possible with **n** number of nodes is $\frac{1}{n+1} 2n_{c_n}$
8. The maximum and minimum size that an array may require to store a binary tree with ***n*** number of nodes are:

   Maximum size = $2^n - 1$

   Minimum size = $2^{\lceil \log_2(n+1) \rceil} - 1$
9. In a linked list representation of binary tree, if there are ***n*** number of nodes, then the number of *NULL* link are ***λ = n +1.***

Representation of Binary Tree

Binary tree can be represented in two ways.

1. Linear (or) Sequential representation using arrays.
2. Linked List representation using pointers.

1. Linear (or) Sequential representation using arrays

- In this representation, a block of memory for an array is to be allocated before going to store the actual tree in it.
- Once the memory is allocated, the size of the tree will be restricted to memory allocated.
- In this representation, the nodes are stored level by level starting from zero level, where only *ROOT* node is present.
- The *ROOT* node is stored in the first memory location. i.e. first element in the array.

The following rules are used to decide the location on any node of tree in the array. (Assume the array index start from 1)

1. The ROOT node is at index**1**.
2. For any node with index i, $1 \leq i \leq n$.

   (a) **Parent (i)** = $\left\lfloor \frac{i}{2} \right\rfloor$

   For the node when i = 1, there is ***no parent node***.

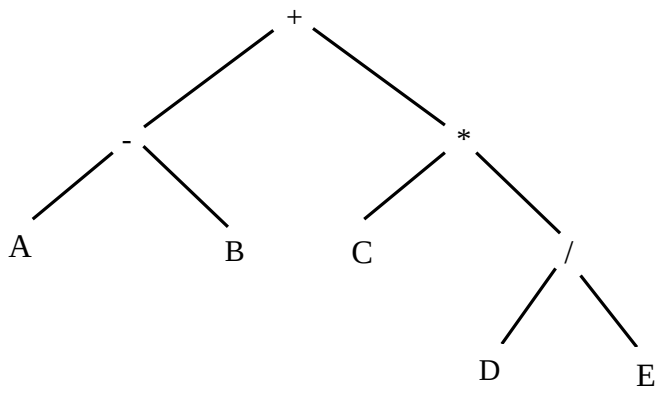
   (b) **LCHILD (i)** = 2 * i

   If 2 * i > n then **i** has ***no left child***.

   (c) **RCHILD (i)** = 2*i +1

   If 2*i + 1 > n then **i** has ***no right child***.

Eg. (A-B) + C * (D / E)



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+	-	*	A	B	C	/							D	E

Array representation of above binary tree.

Advantages of Linear representation of Binary Tree

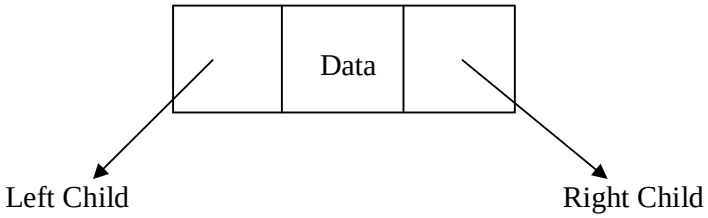
1. Any node can be accessed from any other node by calculating the index and this is efficient from execution point of view.
2. Here only data is stored without any pointers to their successor (or) predecessor.

Disadvantages of Linear representation of Binary Tree

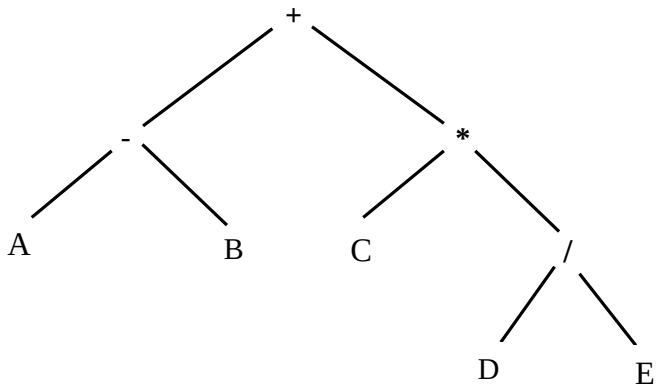
1. Other than full binary tree, majority of entries may be empty.
2. It allows only static memory allocation.

2. Linked List representation of Binary Tree using pointers

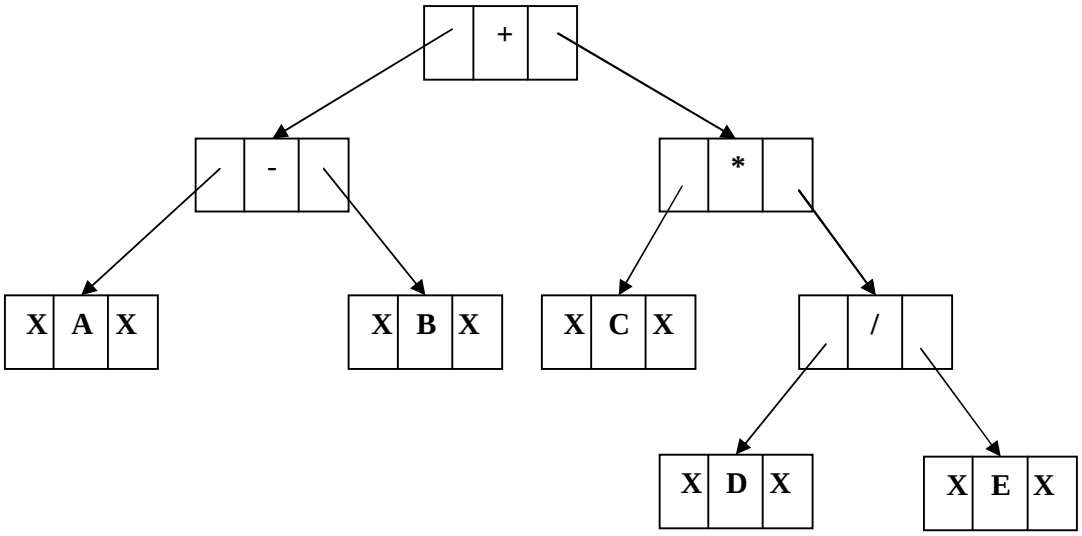
- Linked list representation of assumes structure of a node as shown in the following figure.



- With linked list representation, if one knows the address of *ROOT* node, then any other node can be accessed.



Binary Tree



Linked List representation of Binary Tree

Advantages of Linked List representation of Binary Tree

1. It allows dynamic memory allocation.
2. We can overcome the drawbacks of linear representation.

Disadvantages of Linked List representation of Binary Tree

1. It requires more memory than linear representation. i.e. Linked list representation requires extra memory to maintain pointers.

Binary Tree Traversals

Traversal operation is used to visit each node present in binary tree exactly once.

A Binary tree can be traversed in 3 ways.

1. Preorder traversal
2. Inorder traversal
3. Postorder traversal

1. Preorder traversal

Here first **ROOT** node is visited, then **LEFT** sub tree is visited in *Preorder* fashion and then **RIGHT** sub tree is visited in *Preorder* fashion.

i.e. *ROOT*, LEFT, RIGHT (or) ROOT, RIGHT, LEFT

2. Inorder traversal

Here first **LEFT** subtree is visited in *inorder* fashion, then **ROOT** node is visited and then **RIGHT** sub tree is visited in *inorder* fashion.

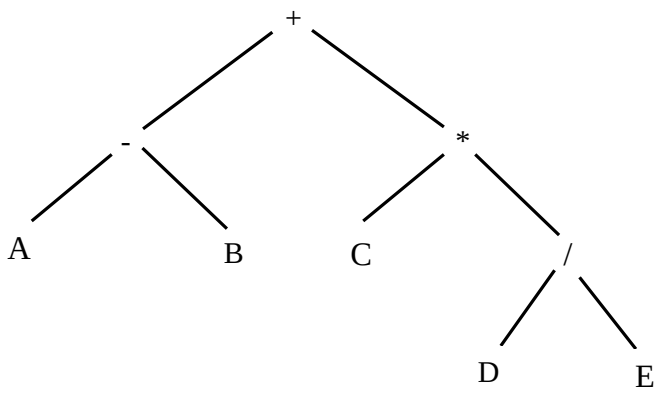
i.e. LEFT, *ROOT*, RIGHT (or) RIGHT, *ROOT*, LEFT

3. Postorder traversal

Here first **LEFT** subtree is visited in *postorder* fashion, then **RIGHT** sub tree is visited in *postorder* fashion and then **ROOT** node is visited.

i.e. LEFT, RIGHT, *ROOT* (or) RIGHT, LEFT, *ROOT*

Eg:



Preorder traversal : + - A B * C / D E

Inorder traversal : A – B + C * D / E

Postorder traversal : A B – C D E / * +

Recursive Binary Tree Traversals

Algorithm preorder(ptr)

Input: Binary Tree with some nodes.

Output: *preorder* traversal of given Binary Tree.

1. if(ptr != NULL)
    a) print(ptr.data)
    b) preorder(ptr.lchild)
    c) preorder(ptr.rchild)
2. end if

End preorder

Algorithm inorder(ptr)

Input: Binary Tree with some nodes.

Output: *inorder* traversal of given Binary Tree.

1. if(ptr != NULL)

a) inorder(ptr.lchild)

b) print(ptr.data)

c) inorder(ptr.rchild)

2. end if

End inorder

Algorithm postorder(ptr)

Input: Binary Tree with some nodes.

Output: *postorder* traversal of given Binary Tree.

1. if(ptr != NULL)

a) postorder(ptr.lchild)

b) postorder(ptr.rchild)

c) print(ptr.data)

2. end if

End postorder

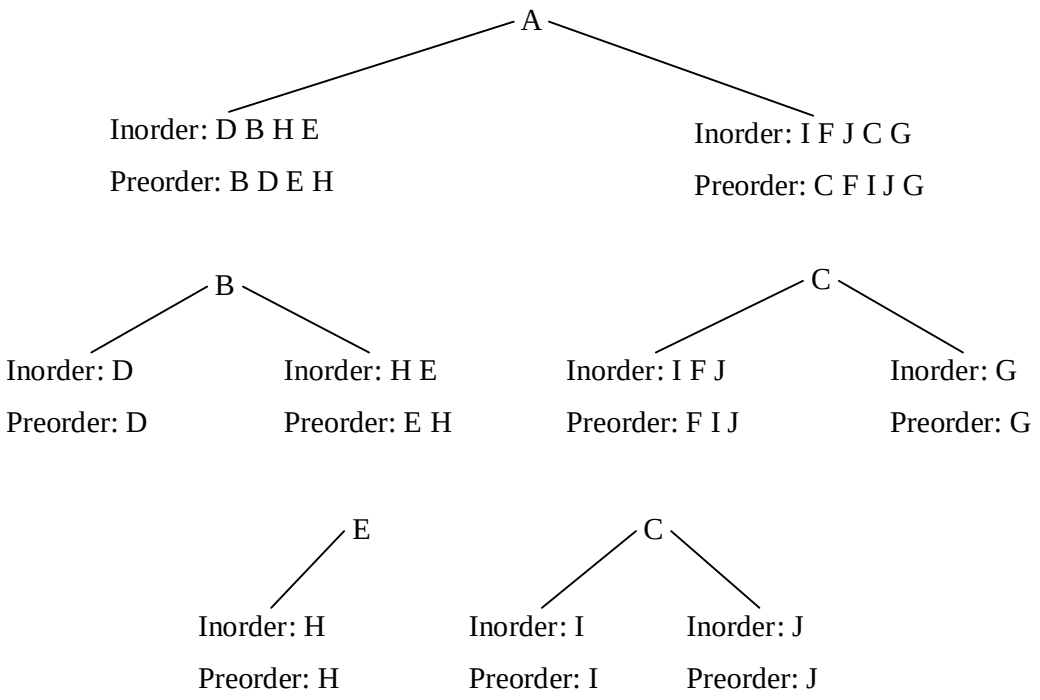
Creation of Binary Tree from its Tree traversals

- A binary tree can be constructed from its traversals.
- If the *Preorder* traversals is given, then the ***first node*** is *ROOT* node and *Postorder* traversal is given then ***last node*** is the *ROOT* node.
- For construction of a binary tree from its traversals, two traversals are essentials. Out of which one should be *inorder* traversal and another one is either *preorder* (or) *postorder* traversal.
- If preorder and post order is given to construct a binary tree, then binary tree can't be obtain *uniquely*.

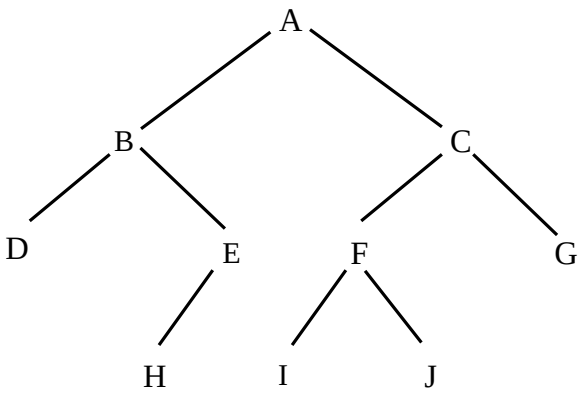
Eg.1.

Inorder:	D	B	H	E	A	I	F	J	C	G
Preorder:	A	B	D	E	H	C	F	I	J	G

1. From the preorder traversal, *A* is the *ROOT* node.
2. In the inorder traversal, all the nodes which are LEFT side of **A** belongs to LEFT sub tree and those node which are RIGHT side of **A** belongs to RIGHT sub tree.
3. Now the problem is reduced to two sub trees and same procedure can be applied repeatedly.

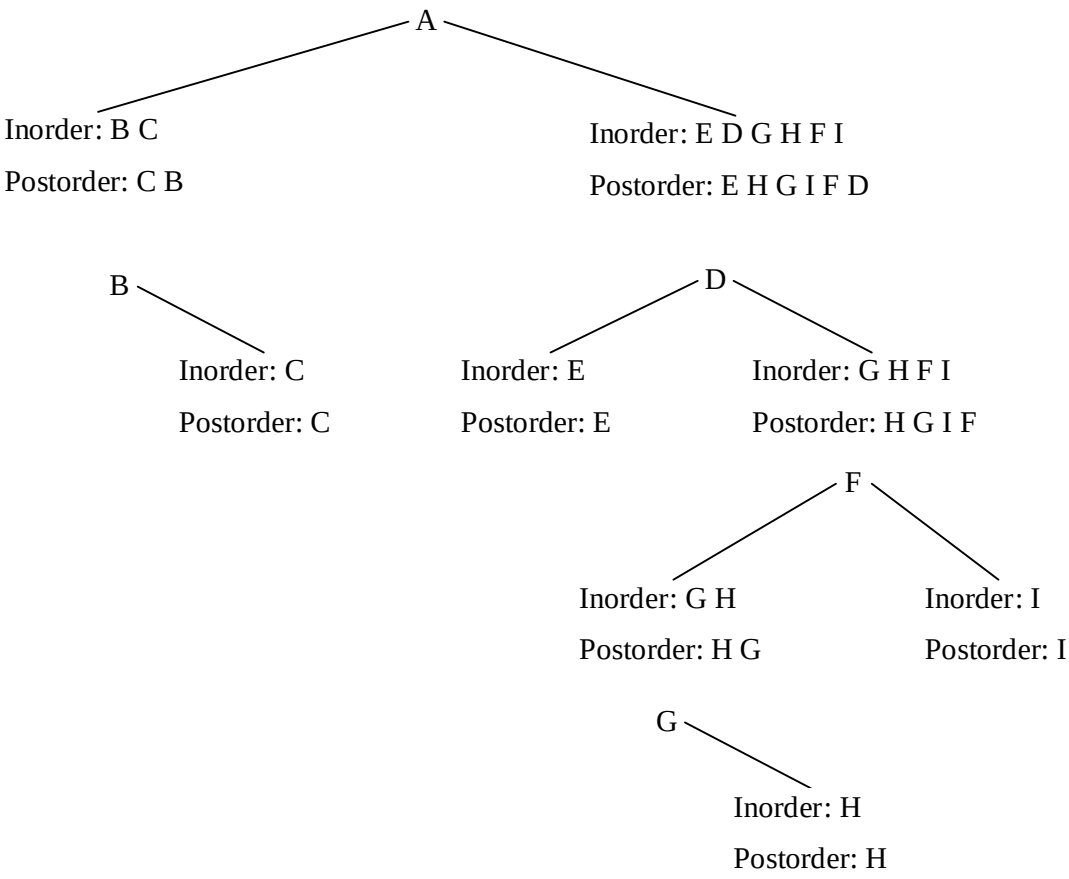


Final Binary tree from the Inorder and Preorder as follows:

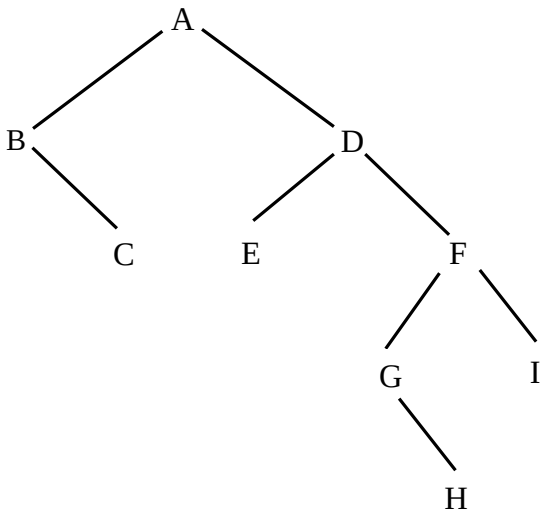


Eg. 2.

Inorder:	B	C	A	E	D	G	H	F	I
Postorder:	C	B	E	H	G	I	F	D	A



Final Binary tree from the Inorder and Postorder as follows:



Types of Binary Tress

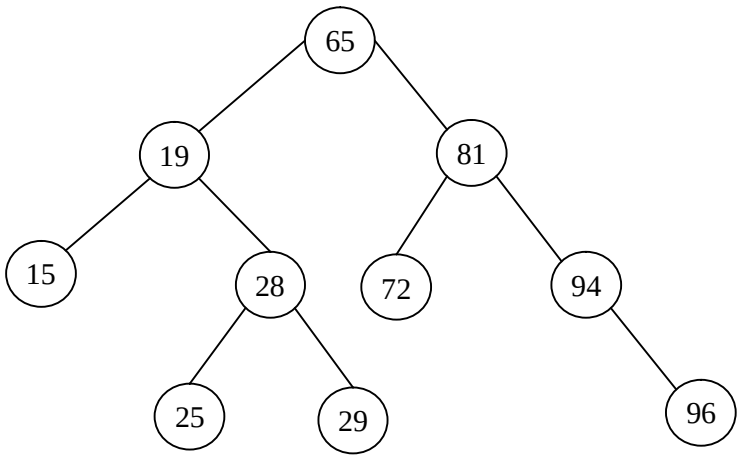
1. Expression Trees
2. Binary Search Trees
3. Threaded Binary Trees
4. Heap Tree
5. Height Balanced Binary Trees
6. Decision Trees
7. Huffman Tree

Binary Search Tree

Definition:

A binary tree T is termed binary search tree (or binary sorted tree) if each node N of T satisfies the following property:

The value at N is greater than every value in the left sub-tree of N and is less than every value in the right sub-tree of N.



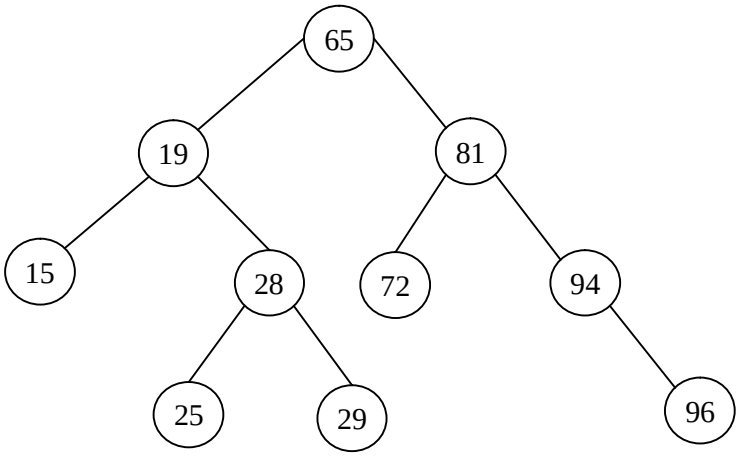
Binary Search Tree with numeric data

Operations on Binary Search Trees:

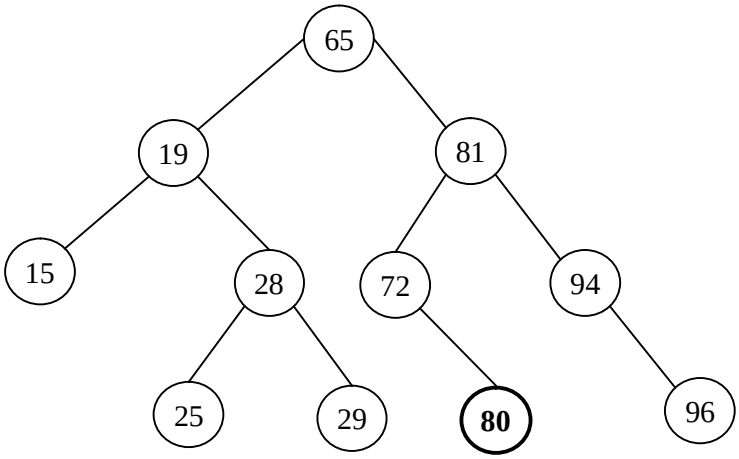
1. Inserting data
2. Deleting Data
3. Traversing the Tree

1. Inserting data into a binary search tree

To insert a node with data say item into a binary search tree, first binary search tree is searched starting from ROOT node for the item. If the item is found, do nothing. Otherwise item is to be inserted as LEAF node where search is halt.



Inserting 80 into the above figure



After insertion of new node 80

Algorithm BST_Insert(item)

Input: *item* is data part of new node to be insert into BST.

Output: BST with new node has data part *item.*

1. ptr = Root
2. flag = 0
3. while(ptr != NULL && flag == 0)
    a) if(item == ptr.data)
        i) flag = 1
        ii) print “item already exist”
    b) else if(item < ptr.data)
        i) ptr1 = ptr
        ii) ptr = ptr.LCHILD

c) else if(item > ptr.data)

i) ptr1 = ptr

ii) ptr = ptr.RCHILD

d) end if

4. end loop

5. if(ptr == NULL)

a) new = getnewnode()

b) new.data = item

c) new.lchild = NULL

d) new.rchild = NULL

e) if(root.data == NULL)

i) root = new

A) print “New node inserted successfully as ROOT Node”

f) else if(item < ptr1.data) /* inserting new node as left child to its parent*/

i) ptr1.lchild = new

ii) print “New Node is inserted successfully as LEFT child”

g) else /* inserting new node as right child to its parent*/

i) ptr1.rchild = new;

ii) print “New Node is inserted successfully as Right Child”

h) end if

6. end if

End BST_Insert

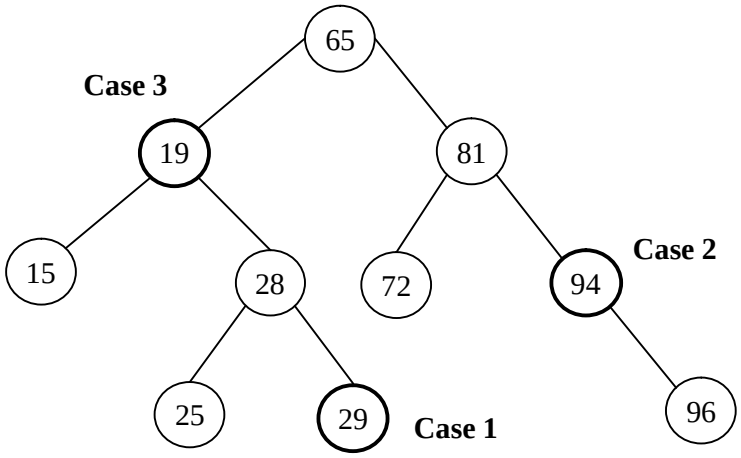
2. Deleting data from a Binary Search Tree

- If ITEM is the information given which is to be deleted from a BST. Let N be the node which contains the information ITEM. Assume PARENT(N) denotes the parent node of N and SUCC(N) denotes the inorder successor of N.
- Then the deletion of the node N depends on the number of its children. Hence, 3 cases may arise and they are:

Case 1: N is the leaf node. i.e. no child nodes.

Case 2: N has exactly one child.

Case 3: N has two children.

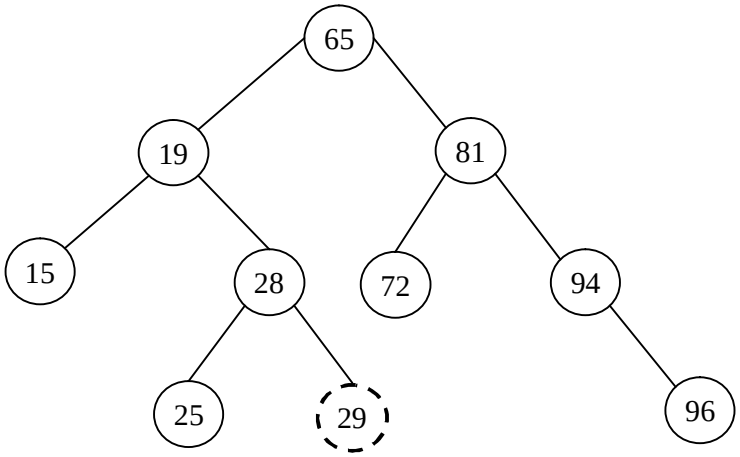


Three cases from deleting a node from BST

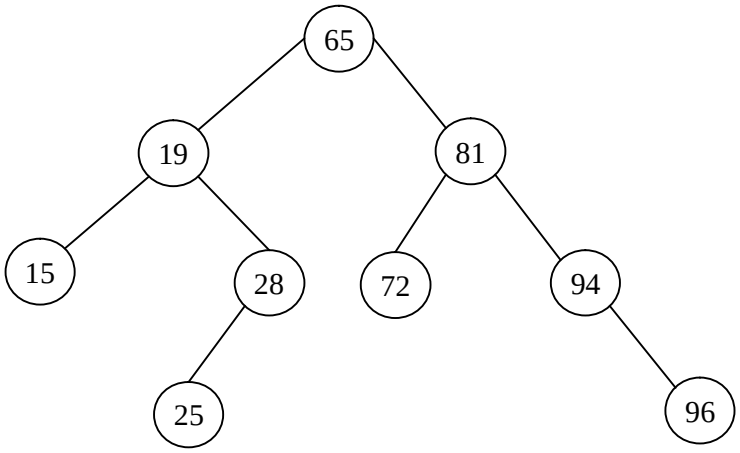
In the above Binary Search Tree (BST) deletion of a node *29* leads to *Case 1*. Here node *29* is a leaf node i.e. which does not have any child nodes. Deletion of node *94* leads to *Case 2*. Here node *94* has only one child node. Deletion of a node *19* leads to *Case 3*. Here node 19 has two child nodes.

Case 1:

N is a leaf node, this node is to be delete. N is deleted from T by simply setting the pointer of N in the parent node PARENT(N) by **NULL** value.



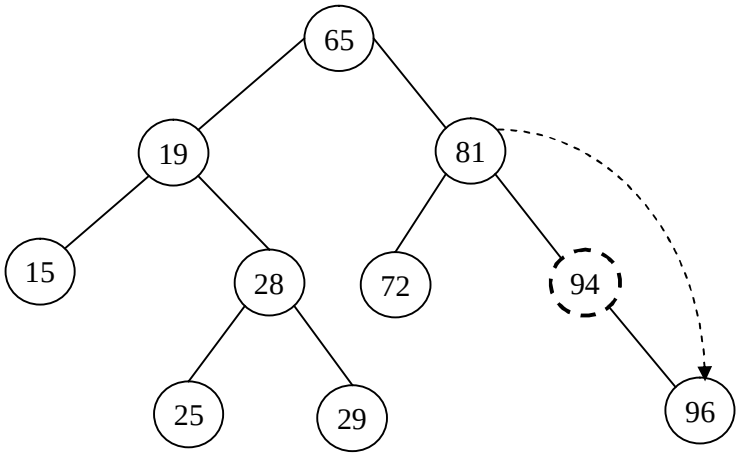
Deletion of node 29



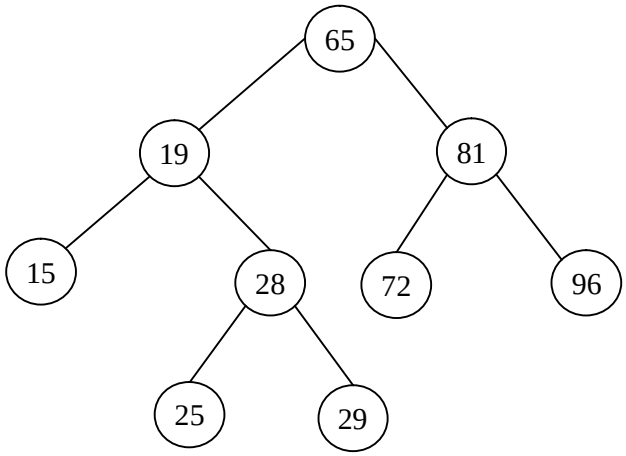
After deletion of node 29 form given BST

Case 2: N has exactly one child.

N is deleted from T by simply replacing the pointer of N in PARENT(N) by the pointer of the only child of N.



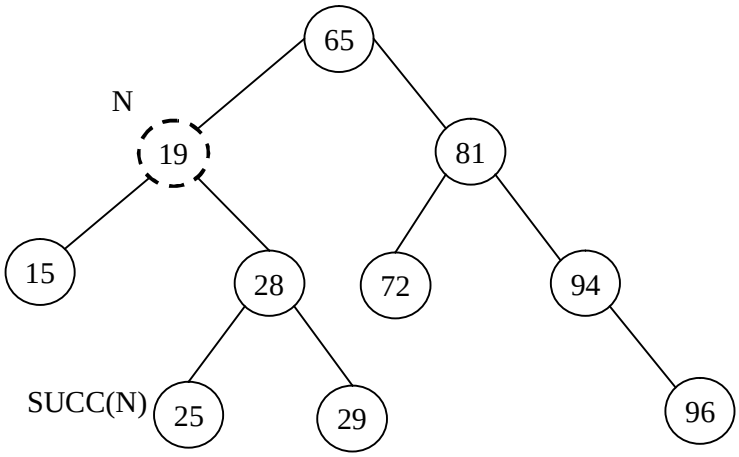
Deletion of node 94



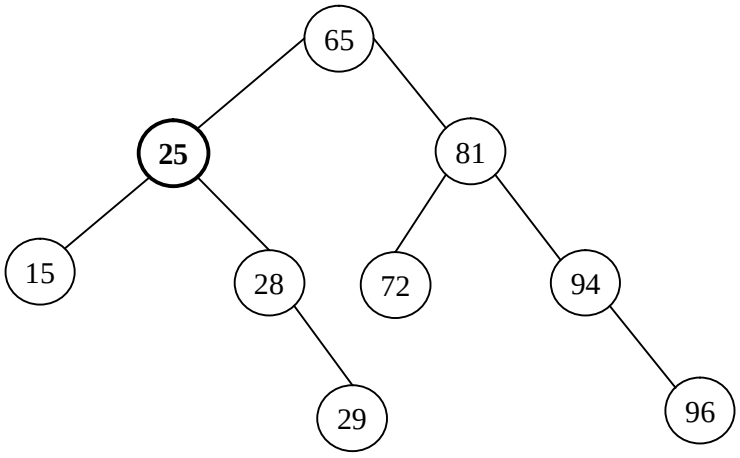
After deletion of node 94 form given BST

Case 3: N has two child nodes.

N is deleted from T by first deleting SUCC(N) from T(by using Case 1 or Case 2 it can be verified that SUCC(N) never has a left child) and then replacing the data content in node N by the data content in node SUCC(N).



Deletion of node 19



After deletion of node 19 form given BST

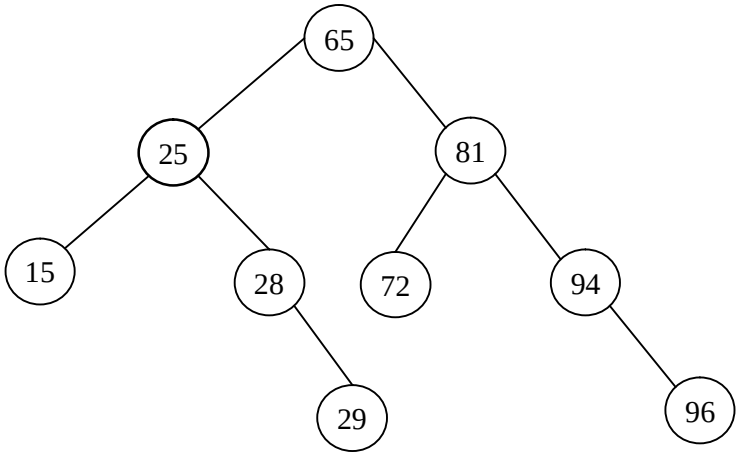
3. Binary Search Tree Traversals

Traversal operation is used to visit each node present in binary search tree exactly once.

A Binary search tree can be traversed in 3 ways.

1. Preorder traversal
2. Inorder traversal
3. Postorder traversal

Example:



Inorder traversal for above BST is: 15, 25, 28, 29, 65, 72, 81, 94, 96

(Inorder traversal of BST always gives Ascending order of elements)

Preorder traversal for above BST is: 65, 25, 15, 28, 29, 81, 72, 94, 96

Postorder traversal for above BST is: 15, 29, 28, 25, 72, 96, 94, 81, 65

Recursive Binary Search Tree Traversals

Same as Recursive implementation of Binary Tree Traversals

Algorithm preorder(ptr)

Input: Binary Tree with some nodes.

Output: *preorder* traversal of given Binary Tree.

1. if(ptr != NULL)
    a) print(ptr.data)
    b) preorder(ptr.lchild)
    c) preorder(ptr.rchild)
2. end if

End preorder

Algorithm inorder(ptr)

Input: Binary Tree with some nodes.

Output: *inorder* traversal of given Binary Tree.

1. if(ptr != NULL)
    a) inorder(ptr.lchild)
    b) print(ptr.data)
    c) inorder(ptr.rchild)
2. end if

End inorder

Algorithm postorder(ptr)

Input: Binary Tree with some nodes.

Output: *postorder* traversal of given Binary Tree.

1. if(ptr != NULL)
    a) postorder(ptr.lchild)
    b) postorder(ptr.rchild)
    c) print(ptr.data)
2. end if

End postorder

Creation of Binary Search Tree from its Tree traversals

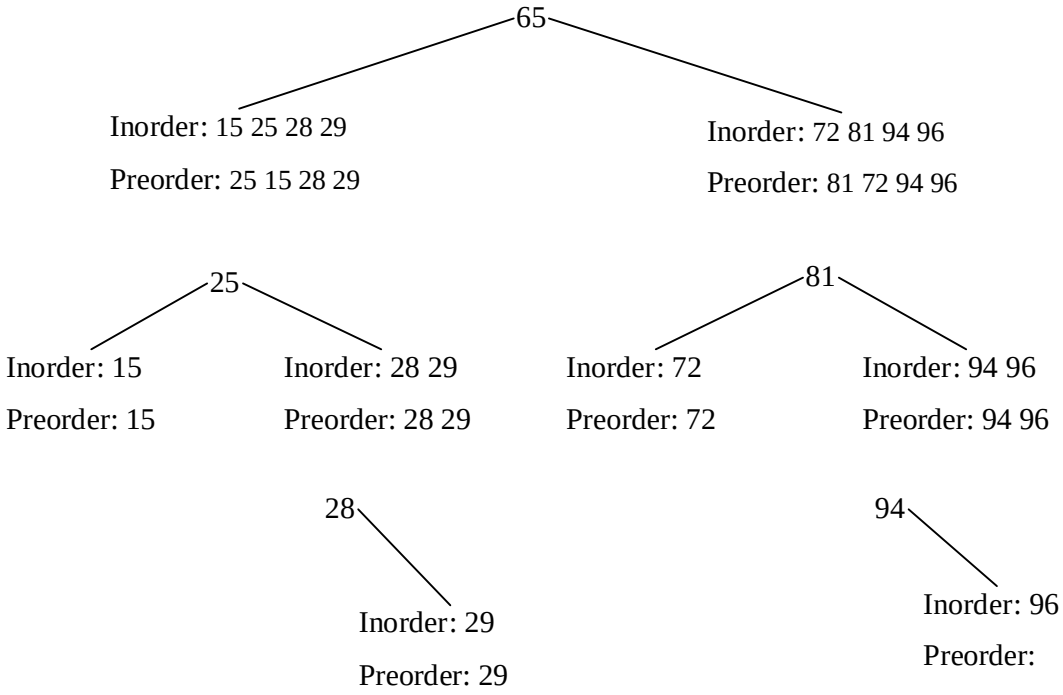
- A binary search tree can be constructed from its traversals.
- If the *Preorder* traversals is given, then the ***first node*** is *ROOT* node and *Postorder* traversal is given then ***last node*** is the *ROOT* node.

- For construction of a binary search tree from its traversals, two traversals are essentials. Out of which one should be *inorder* traversal and another one is either *preorder* (or) *postorder* traversal.

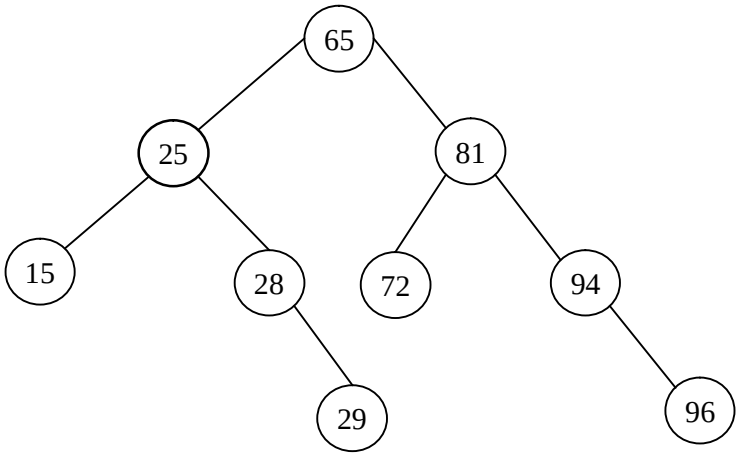
Eg.1:

Inorder:	15	25	28	29	65	72	81	94	96
Preorder:	65	25	15	28	29	81	72	94	96

1. From the preorder traversal, *65* is the *ROOT* node.
2. In the inorder traversal, all the nodes which are LEFT side of **65** belongs to LEFT sub tree and those node which are RIGHT side of **65** belongs to RIGHT sub tree.
3. Now the problem is reduced to two sub trees and same procedure can be applied repeatedly.



Final Binary Search Tree from the Inorder and Preorder as follows:


65
25
81
15
28
72
94
29
96

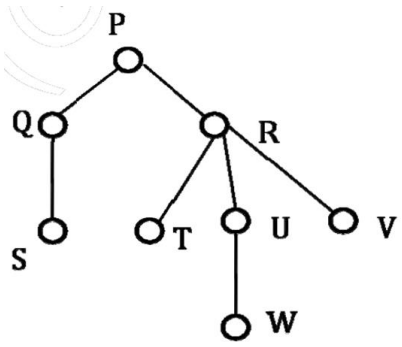
UNIT-IV
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. How many nodes in a tree have ***no*** ancestors? [ ]
   (A) 0 (B) 1 (C) 2 (D) n
2. What is the maximum possible number of nodes in a binary tree at level ***6***? [ ]
   (A) 6 (B) 12 (C) 64 (D) 32
3. A full binary tree with *2n+1* nodes contain_____? [ ]
   (A) n leaf nodes (B) n non-leaf nodes
   (C) n-1 leaf nodes (D) n-1 non-leaf nodes
4. A full binary tree with *n* leaves contains_____? [ ]
   (A) n nodes (B) log_2^n nodes (C) 2n –1 nodes (D) 2^n nodes
5. The number of leaf nodes in a complete binary tree of depth d is _____? [ ]
   (A) 2^d (B) $2^{d-1}+1$ (C) $2^{d+1}+1$ (D) 2^d+1
6. The pre-order and post order traversal of a Binary Tree generates the same output. The tree can have maximum_____. [ ]
   (A) Three nodes (B) Two nodes
   (C) One node (D) Any number of nodes
7. The height of a tree is the length of the longest root-to-leaf path in it. The maximum and minimum number of nodes in a binary tree of height 5 are: _____ [ ]
   (A) 63 and 6, respectively (B) 64 and 5, respectively
   (C) 32 and 6, respectively (D) 31 and 5, respectively
8. If a node in a BST has two children, then its *inorder predecessor* has _____. [ ]
   (A) No left child (B) No right child
   (C) Two children (D) No child
9. In order to get the contents of a Binary search tree in ascending order, one has to traverse it in _____ fashion? [ ]
   (A) pre-order (B) in-order (C) post order (D) Not possible
10. A BST is traversed in the following order recursively: **right, root, left.** The output sequence will be in_____. [ ]
    (A) Ascending order (B) Descending order
    (C) Bitomic sequence (D) No specific order
11. In order to get the information stored in a Binary Search Tree in the descending order, one should traverse it in which of the following order? [ ]
    (A) left, root, right (B) root, left, right

(C) right, root, left (D) right, left, root

12. What is common in three different types of traversals (Inorder, Preorder and Postorder)?
    (A) Root is visited before right subtree
    (B) Left subtree is always visited before right subtree
    (C) Root is visited after left subtree
    (D) All of the above
13. A binary search tree contains the numbers 1, 2, 3, 4, 5, 6, 7, 8. When the tree is traversed in pre-order and the values in each node printed out, the sequence of values obtained is 5, 3, 1, 2, 4, 6, 8, 7. If tree is traversed in post-order, the sequence obtained would be____ [ ]
    (A) 8, 7, 6, 5, 4, 3, 2, 1 (B) 1, 2, 3, 4, 8, 7, 6, 5
    (C) 2, 1, 4, 3, 6, 7, 8, 5 (D) 2, 1, 4, 3, 7, 8, 6, 5
14. Suppose that we have numbers between 1 and 100 in a binary search tree and want to search for the number 55. Which of the following sequences CANNOT be the sequence of nodes examined? [ ]
    (A) {10, 75, 64, 43, 60, 57, 55} (B) {90, 12, 68, 34, 62, 45, 55}
    (C) {9, 85, 47, 68, 43, 57, 55} (D) {79, 14, 72, 56, 16, 53, 55}
15. Consider the following rooted tree with the vertex P labeled as root. The order in which the nodes are visited during in-order traversal is_____? [ ]
    (A) SQPTRWUV (B) SQPTURWV (C) SQPTWUVR (D) SQPTRUWV



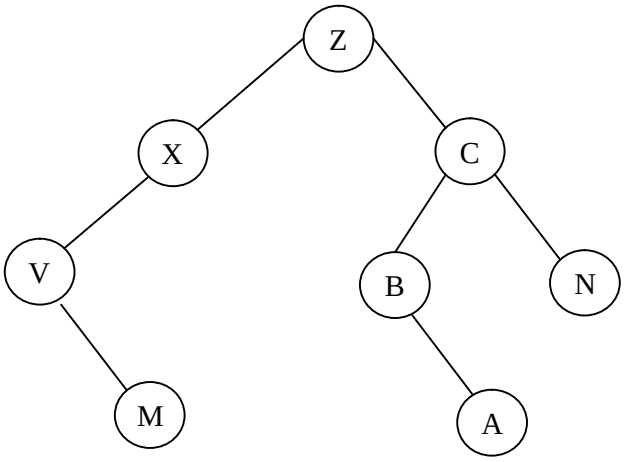
Explanation:

The only confusion in this question is, there are 3 children of R. So when should R appear – after U or after T? There are two possibilities: SQPTRWUV and SQPTWURV. Only 1st possibility is present as an option A, the 2nd possibility is not there. Therefore option A is the right answer.

SECTION-B

Descriptive Questions

1. Write recursive algorithms for Binary Search Tree Traversals.
2. What is the inorder, preorder and postorder for the following binary tree?



3. Construct Binary Tree for the following tree traversals.

   Inorder: W U R O P I T Y E

   Preorder: P O U W R I Y T E

   What is the **Post order** traversal for the above constructed binary tree?

Ans: W R U O T E Y I P

4. Construct Binary Tree for the following tree traversals.

   Inorder: N Z V A M C B S X D

   Postorder: Z A V N C S D X B M

   What is the **Preorder** traversal for the above constructed binary tree?

Ans: M N V Z A B C X S D

5. Create Binary Search with the following elements.

20 30 15 25 42 61 72 18 10 8

What is the **Inorder** traversal for the above constructed Binary Search tree?

Ans: 8 10 15 18 20 25 30 42 61 72

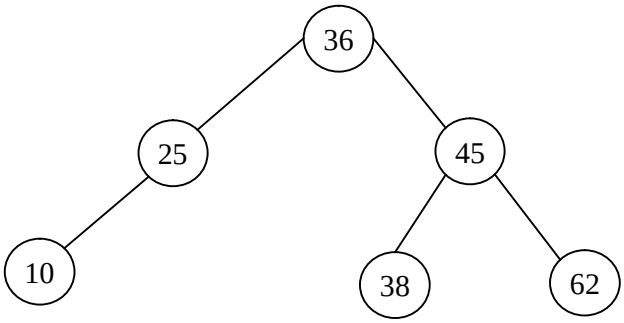
6. Create Binary Search with the following elements.

100 90 110 80 95 125 115 108 104 76 49 62

What is the **Inorder** traversal for the above constructed Binary Search tree?

Ans: 49 62 76 80 90 95 100 104 108 110 115 125

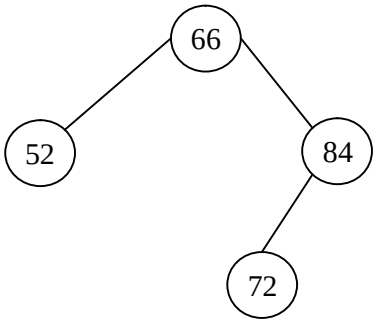
7. Consider the following Binary Search Tree and perform the following sequence of operations.



Insert the elements 55, 68, 49, 18, 28, 27, 30. Now **delete** the elements 55, 45, 36, 10 and 18. Finally what is the root node?

Ans: 38

8. Consider the following Binary Search Tree and perform the following sequence of operations.



Insert the elements 89, 46, 48, 26, 76, 98, 100. Now **delete** the elements 84, 48, 52 and 66. Finally what is the root node?

Ans: 72

Section - C

1. Consider a binary tree T that has 200 leaf nodes. Then, the number of nodes in T that have exactly two children are? **(GATE 2016)** **[ ]**

   (A) 201 (B) 100 (C) 199 (D) 50

2. The maximum number of binary trees that can be formed with three unlabelled nodes is: _____ **(GATE 2007)** [ ]

   (A) 1 (B) 5 (C) 4 (D) 3

3. The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is: **(GATE 2007)** [ ]

   (A) 2^h -1 (B) 2^(h-1) – 1 (C) 2^(h+1) -1 (D) 2*(h+1)

4. The inorder and preorder traversal of a binary tree are ***d b e a f c g*** and ***a b d e c f g***, respectively. The postorder traversal of the binary tree is: **(GATE 2007)** [ ]

   (A) d e b f g c a (B) e d b g f c a (C) e d b f g c a (D) d e f g b c a

5. Consider the label sequences obtained by the following pairs of traversals on a labelled binary tree. Which of these pairs identify a tree uniquely? **(GATE CS 2004)** [ ]

   i) preorder and postorder

   ii) inorder and postorder

   iii) preorder and inorder

   iv) level order and postorder

   (A) (i) only (B) (ii), (iii) only (C) (iii) only (D) (iv) only

6. Let **LASTPOST**, **LASTIN** and **LASTPRE** denote the last vertex visited in a postorder, inorder and preorder traversal. Respectively, of a complete binary tree. Which of the following is always true? **(GATE CS 2000)** [ ]

   (A) LASTIN = LASTPOST (B) LASTIN = LASTPRE

   (C) LASTPRE = LASTPOST (D) None of the above

7. While inserting the elements 71, 65, 84, 69, 67, 83 in an empty binary search tree (BST) in the sequence shown, the element in the lowest level is? **(GATE 2015)** **[ ]**

   (A) 65 (B) 67 (C) 69 (D) 83

8. Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty *binary search tree*. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree? **(GATE CS 2003)** [ ]

   (A) 7 5 1 0 3 2 4 6 8 9 (B) 0 2 4 3 1 6 5 9 8 7

   (C) 0 1 2 3 4 5 6 7 8 9 (D) 9 8 6 4 2 3 0 1 5 7

9. Which of the following is/are *correct* inorder traversal sequence(s) of binary search tree(s)? **(GATE 2016)** **[ ]**

   I. 3, 5, 7, 8, 15, 19, 25

   II. 5, 8, 9, 12, 10, 15, 25

   III. 2, 7, 10, 8, 14, 16, 20

   IV. 4, 6, 7, 9 18, 20, 25

   (A) I and IV only (B) II and III only (C) II and IV only (D) II only

10. Postorder traversal of a given binary search tree, T produces the following sequence of keys ***10, 9, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29.*** Which one of the following sequences of keys can be the result of an in-order traversal of the tree T? **(GATE CS 2004)** [ ]

    (A) 9, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

    (B) 9, 10, 15, 22, 40, 50, 60, 95, 23, 25, 27, 29

    (C) 29, 15, 9, 10, 25, 22, 23, 27, 40, 60, 50, 95

    (D) 95, 50, 60, 40, 27, 23, 22, 25, 10, 9, 15, 29

11. The following numbers are inserted into an *empty binary search tree* in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)? **(GATE CS 2004)** [ ]

    (A) 2 (B) 3 (C) 4 (D) 6

12. The *preorder* traversal sequence of a *binary search tree* is 30, 20, 10, 15, 25, 23, 39, 35, and 42. Which one of the following is the postorder traversal sequence of the same tree?

    (GATE 2013) [ ]

    (A) 10, 20, 15, 23, 25, 35, 42, 39, 30 (B) 15, 10, 25, 23, 20, 42, 35, 39, 30

    (C) 15, 20, 10, 23, 25, 42, 35, 39, 30 (D) 15, 10, 23, 25, 20, 35, 42, 39, 30

13. Let T be a binary search tree with 15 nodes. The minimum and maximum possible heights of T are: **(GATE-CS-2017 -Set 1)** **[ ]**

    Note: The height of a tree with a single node is 0.

    (A) 4 and 15 respectively (B) 3 and 14 respectively

    (B) 4 and 14 respectively (D) 3 and 15 respectively

14. Let T be a tree with 10 vertices. The sum of the degrees of all the vertices in T is _____.

    (GATE-CS-2017 - Set 1) [ ]

    (A) 18 (B) 19 (C) 20 (D) 21

Explanation:

Given, v= Total vertices = 10 e = v - 1 =9 Degree = 2 * e = 18 Therefore, option A is correct.

15. The pre-order traversal of a binary search tree is given by 12, 8, 6, 2, 7, 9, 10, 16, 15, 19, 17, 20. Then the post-order traversal of this tree is: **(GATE-CS-2017 -Set 2)** [ ]

    (A) 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20

    (B) 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12

    (C) 7, 2, 6, 8, 9, 10, 20, 17, 19, 15, 16, 12

    (D) 7, 6, 2, 10, 9, 8, 15, 16, 17, 20, 19, 12