Unit 7 Trees

7.1 Introduction

Tree:

Tree is an important non-linear data structure used to represent a relationship between elements such as records, nodes and table of contents. Trees are used in compiler design and construction, data base design, operating system, windows, evaluation of arithmetic expression, etc.

A tree is a non-empty collection of vertices and edges with some restrictions. A vertex is a node and edge is connection between two nodes.

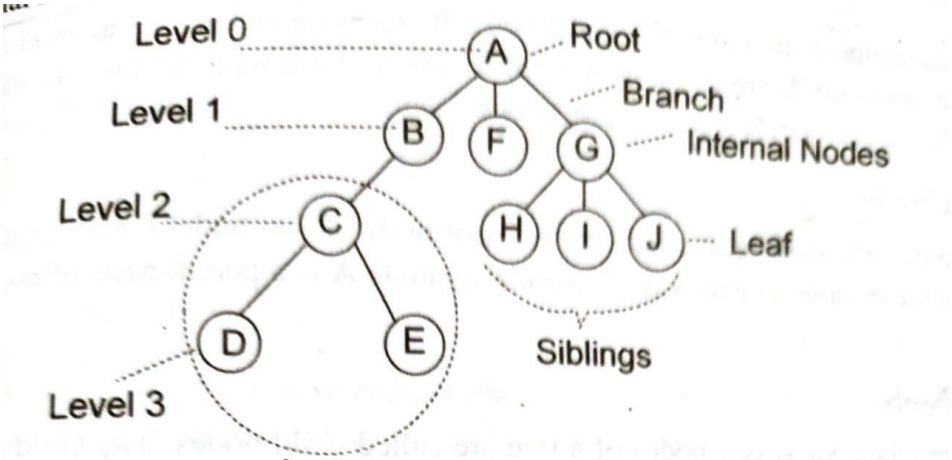
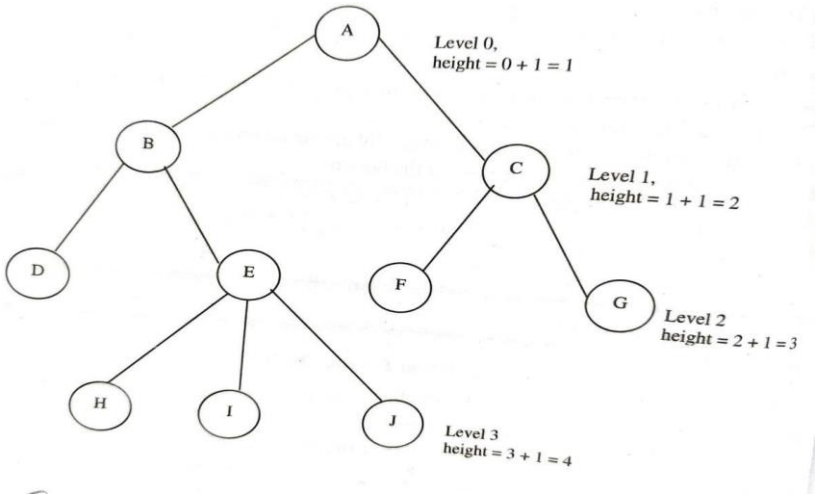


Figure 7.1 A Tree Data structure

In the above figure of a tree,

- Parents: A, B, C and G
- Children: B, F, G, C, H, I, J, D, E
- Siblings: {B, F, G}, {D, E}, {H, I, J}
- Leaves: F,D,E,H,I,J
- Length: 4(Length or height of the tree = Maximum Level +1 = 3+1=4)



Tree Terminologies:

1. **Node:**

Each data elements in a tree is called a node. It is a basic and main important component of any tree structure. It stores the actual information along with the links to other nodes. In above figure, each circle represents a node.

2. Parent Node:

The parent node is the immediate predecessor node of that node. For example, ‘C’ is the parent node of ‘F’and ‘G’. A' is a parent node of ‘B' and ‘C’.

3. Child Node:

The immediate successor nodes of a tree are called child nodes. The child node placed at the left is known as left child and the child node placed at right side is known as right child node of a tree. In above figure, node ‘B’ is the left child of node ‘A’ and node 'C’ is the right child of node ‘A’.

4. Root:

The root node is a special node of a tree. It is the first node of a tree and it has no parent node. There can be only one root node in a tree. In the figure above, node ‘A’ is a root node.

5. Degree of a node:

The degree of a node is defined as the maximum number of child that a node has. For example, in above figure the node ‘E’ has three children and the non-terminal node has two children. So, the degree of node ‘E’ is 3, the degree of other non-terminal node is 2 and the degree of leaf node is 0.

6. Degree of a tree:

The degree of a tree is defined as the maximum number of degree of a node in that tree. For example, in above tree the maximum degree of a node is 3. So, the degree of a tree is also 3.

7. Terminal nodes / Leaf nodes:

The terminal nodes are those nodes that don’t have any children. Terminal nodes are also called the leaf nodes. In the above figure D,F, G,H,I,J are the leaf nodes.

8. Non-terminal nodes:

The non-terminal nodes are those nodes which have at least one child node. In above figure node ‘A’, ‘B’, ‘C’ and ‘E’ are the non-terminal nodes.

9. Siblings:

The child nodes of a given parent node are called Siblings. They are also called the brother of nodes. For example, in above figure node ‘C’ is the siblings of node ‘B’.

10. Level:

The level of a node is the number of edges along the unique path between it and the root. The level of the root is zero. If the node is at level **i** then its child is at level **i+1** and its parent is at level **i-1.** This rule is common for all nodes except the root node.

11 Edge:

It is the connection line between two nodes. The line drawn from one node to another node is called edge. **12.**

Path:

It is the sequence of consecutive edges from the source node to the destination node. In the above figure the path between node ‘A’ and node ’H’ is given by the node pairs. (A,B) , (B,E) and (E,H).

13. Depth:

The maximum level of any node in a given tree is called the depth. In the above tree, the root node ‘A’ has a maximum level. The term **Height** is also used to denote the depth.

14. Forest:

It is a set of disjoint trees. If we remove a root node in the above tree it becomes a forest. The forest with two trees.

7.2 Binary Tree: Construction, Traversal (pre-order, in-order, post-order) **Binary Tree:**

A binary tree is a finite set of data elements which is either empty or consists of a single data or up to two data element. A binary tree is very important and most commonly used nonlinear data structure. The maximum degree of any node is at most two. This means there may be a zero degree or one-degree node and two-degree node.

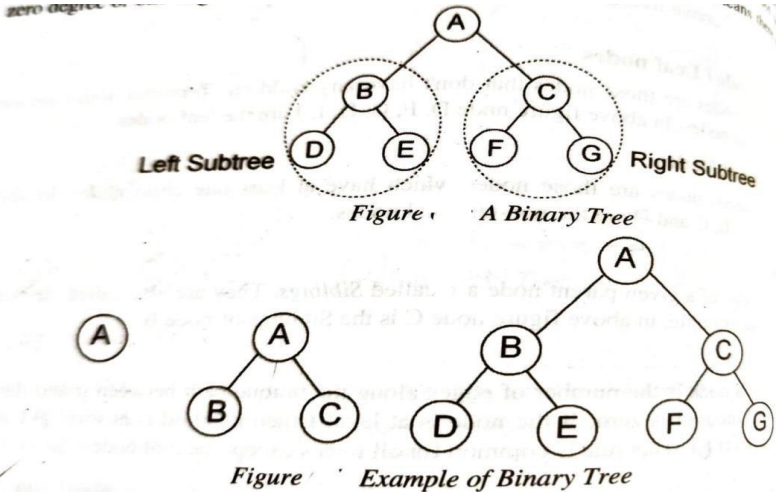


Figure : A Binary Tree

Figure : Example of Binary Tree

Types of Binary Tree

1. Strictly Binary Tree:

A strictly binary tree is one in which every non-terminal nodes consists of nonempty left sub tree and right sub tree, every non-terminal node has a degree of two. Strictly binary tree has either zero or two children.

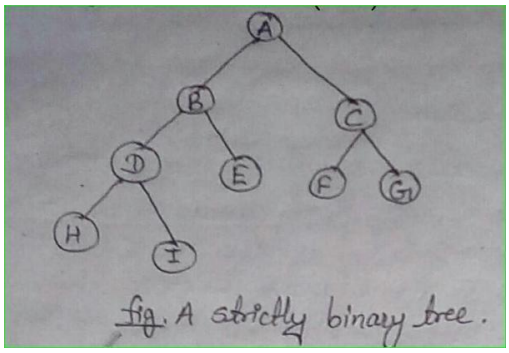


fig. A strictly binary tree.

A strictly binary tree with 'n' leaves always contain (2n-1) nodes. In the above figure, there are 5 leaves and (2*5-1=10-1=9) 9 nodes.

2. Complete Binary Tree:

A binary tree of depth 'd' is called complete binary tree if all of the leaves are at level 'd'. A complete binary tree with depth 'd' has 2^d leaves and 2^d-1 non-leaf nodes. In the following complete binary tree, depth is 3, number of leaf nodes are 2^d i.e 2^3 =8 and non-leaf nodes are 2^d-1 i.e. 2^3-1=81=7.

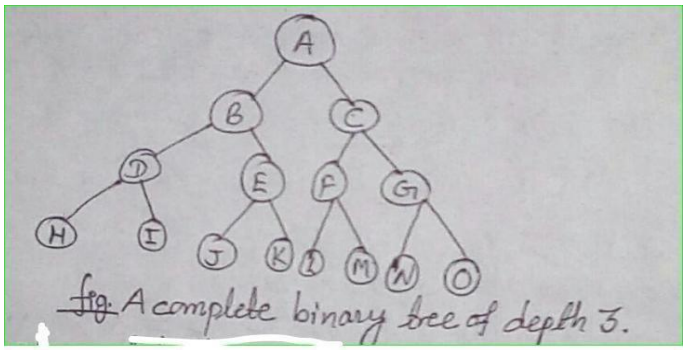


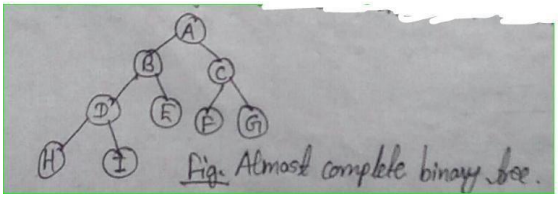
fig. A complete binary tree of depth 3.

1. Almost Complete Binary Tree:

A binary tree of depth 'd' is an almost complete binary tree if

a) Each leaf in the tree is either at level 'd' or at level 'd-1'.

b) Leaves at same level must be filled from left to right.



Binary Tree Construction:

Binary tree can be constructed or created with the help of tree traversals. Using the output of tree traversals we construct a binary tree. There are two different ways of creating or constructing binary tree. They are

1. Pre-order and in-order traversals.
2. Post-order and in-order traversals.

1. Pre-order and in-order traversals

The general procedure for creating a binary tree using pre-order and in-order traversals is as follows.

Step 1: Scan the pre-order traversal from left to right.

Step 2: For each node scanned locate its position in in-order traversal. Let the scanned node be X.

Step 3: The nodes preceding X in in-order form its left sub-tree and nodes succeeding X form the right sub tree.

Step 4: Repeat step 1 for each symbol in the pre-order. Example:

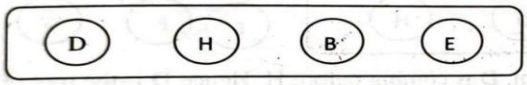
Example 1: Construct the binary tree given the following traversals.
Pre-order: A B D H E C F G In-order: D H B E A F C G

Solution:

(1) In pre-order traversal root is the first node. So, A is the root node of the binary tree, i.e.,

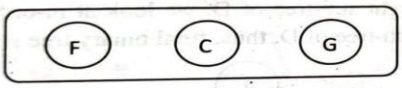
root

Now, we can find the node of left sub-tree and right sub-tree with in-order sequences. Sub-tree and nodes which are in the right side of the root node in in-order sequence are nodes of right sub-tree. So, left and right sub tree will be given by:



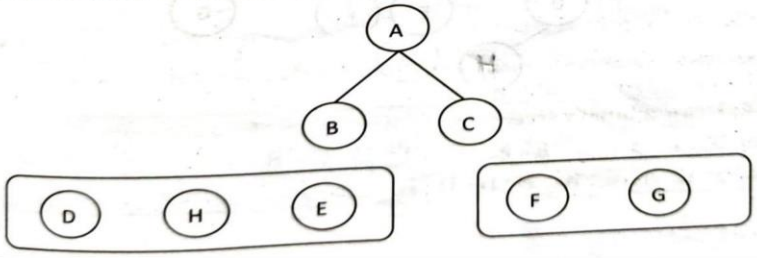
Left sub tree

AND

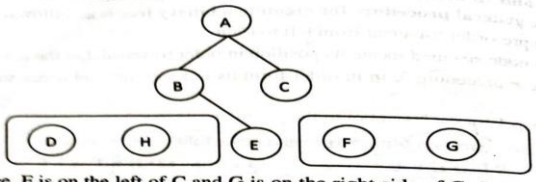


Right sub tree

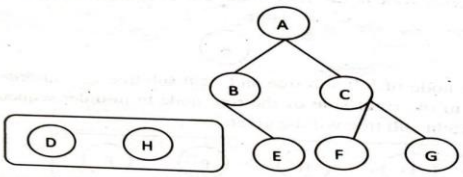
Now, the left child of the root node will be the first node in the pre-order sequence after root node A i.e., B. hence B is the left child of A. Similarly the right child of root A will be the first node after nodes of left sub-tree in preorder sequence. Hence C is the right child of A.



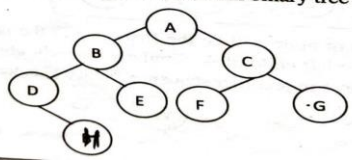
In in-order sequence, D and H are on the left side of B and E is on the right side of B. so, D and H will form left sub-tree and E will be in the right sub-tree of B.



In-order sequence, F is on the left of C and G is on the right side of C. So F will be in left sub tree of C and G in right sub-tree of C.



Again, in pre-order traversal, D is coming before H. Hence D is the root of left sub-tree of B. To find out whether H is in left or right sub-tree of D. we look at in-order sequence. H is in right side of D. So; H will be in right sub-tree of D. thus, final binary tree is:



2. Post-order and in-order traversals.

The general procedure for creating a binary tree using post-order and in-order traversals is as follows.

Step 1: Scan the post-order traversal from right to left.

Step 2: For each node scanned locate its position in in-order traversal. Let the scanned node be X.

Step 3: The nodes preceding X in in-order form its left sub-tree and nodes succeeding X form the right subtree.

Step 4: Repeat step 1 for each symbol in the pre-order. Example:

Example 1: Create a binary tree.

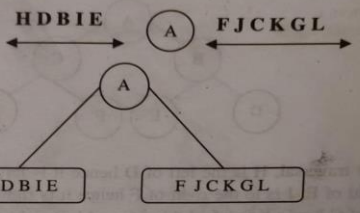
Post-order: H D I E B J F K L G C A

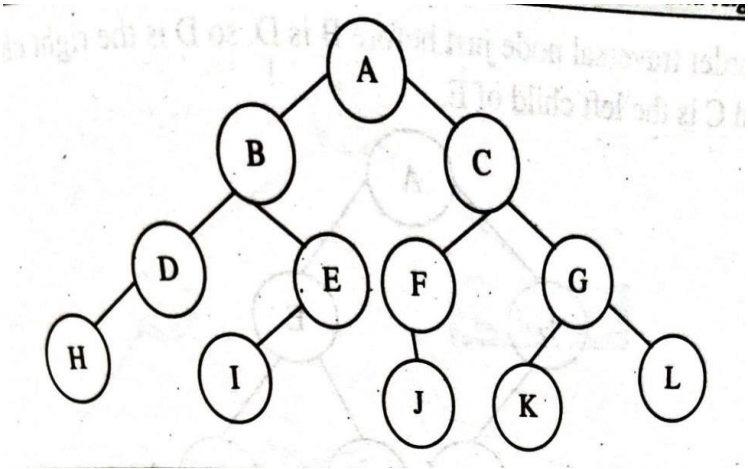
In-order: H D B I E A F J C K G L

Solution:

In post-order traversal root is the last node. Hence A is the root of the binary tree.

Now, from in-order sequence





7.3 Binary Search Tree: Construction, Traversal Binary Search Tree:

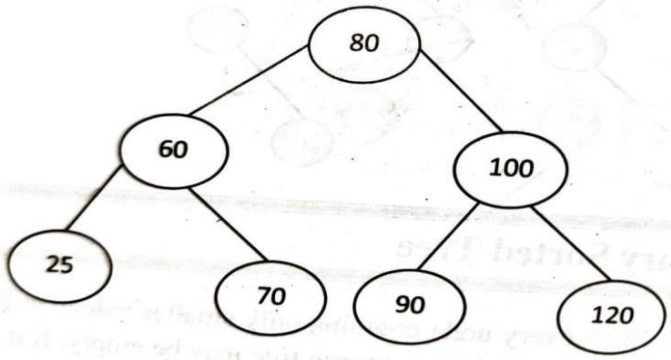
Binary Search Tree (BST) is a binary tree that is either empty or in which every node contains a key (value) and satisfies the following conditions.

a) All keys in the left sub-tree of the root are smaller than the key in the root node.

b) All keys in the right sub-tree are greater than the key in the root node.

c) The left and right sub-trees of the root are again binary search trees.

The following tree is a Binary Search Tree. In this tree, left sub-tree of every node contains nodes with smaller values and right sub-tree of every node contains larger values.

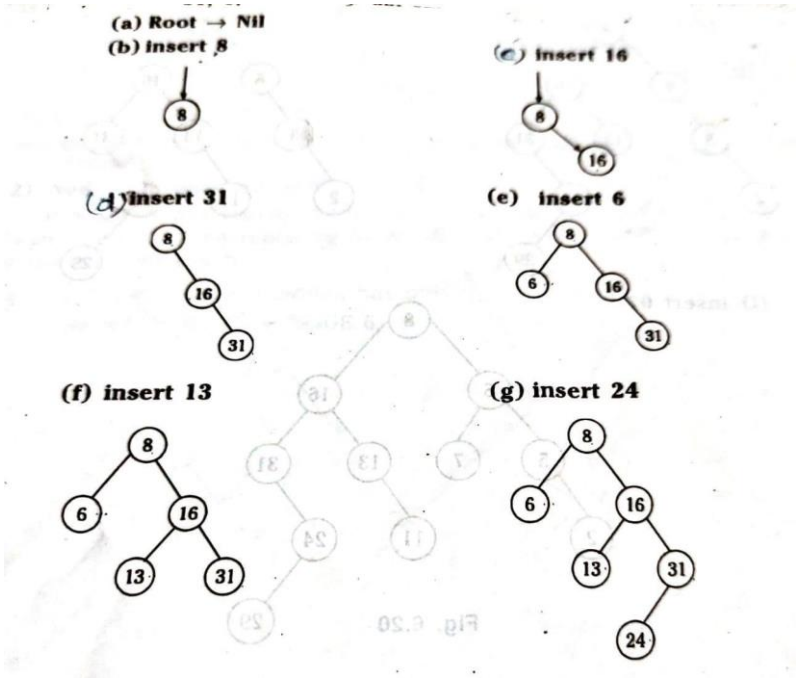


Example:

Construct BST using the following given sequence of numbers.

8,16,31,6,13,24,29,5,2,11,7

Solution:



Operations on Binary Search Tree (BST):

Following operations can be done in BST.

1. Insertion Algorithm:

a) Start from the root node.

b) If the data to be inserted is 'w', compare this with the value of the root node.

i) If they are equal, just stop because this value already exists.

ii) If they are different and if the data to be inserted is less than the value of the root node choose the left sub-tree.

c) Repeat step 'b' until the leaf node is encountered where the data has to be inserted.

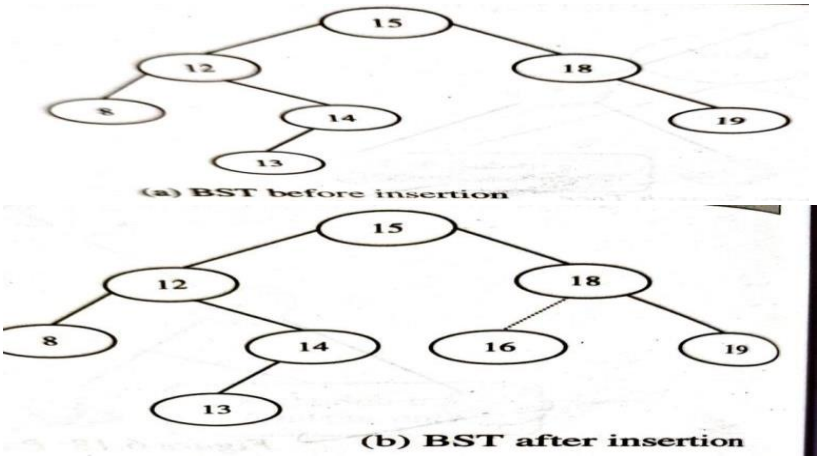
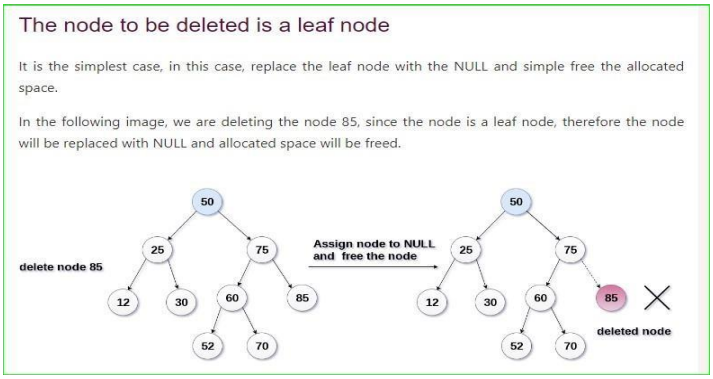


Figure: Insertion of node

In the above figure, if the node to be inserted is 'w' i.e. 16, we compare this node '16' with the root node '15'. Here the node to be inserted is greater than the root node. So, it must be placed on its right subtree. Again the node '16' is compared with the node '18', it is less than the node '18'. So, it must be placed on its left sub-tree according to the properties of Binary Search Tree (BST).

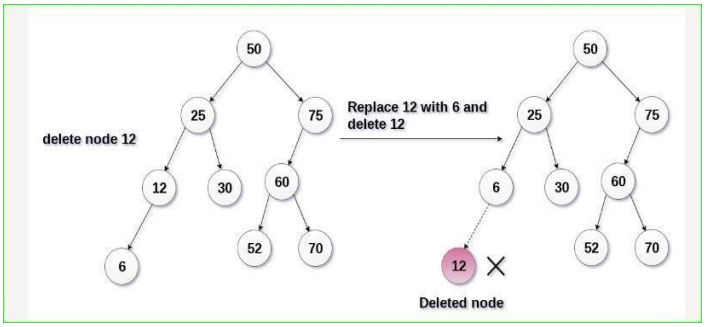
2. Deletion

Deleting node from a binary search tree depends on its position. The position may be one of the following. a) The node to be deleted is a leaf node.



b) The node to be deleted has only one child.

In this case, replace the node with its child and delete the child node which now contains the value which is to be deleted. Simply replace it with the NULL and free the allocated space. In the following image, the node 12 is to be deleted. It has only one child. The node will be replaced with its child node and the replaced node 12 (which is now leaf node) will simply be deleted.



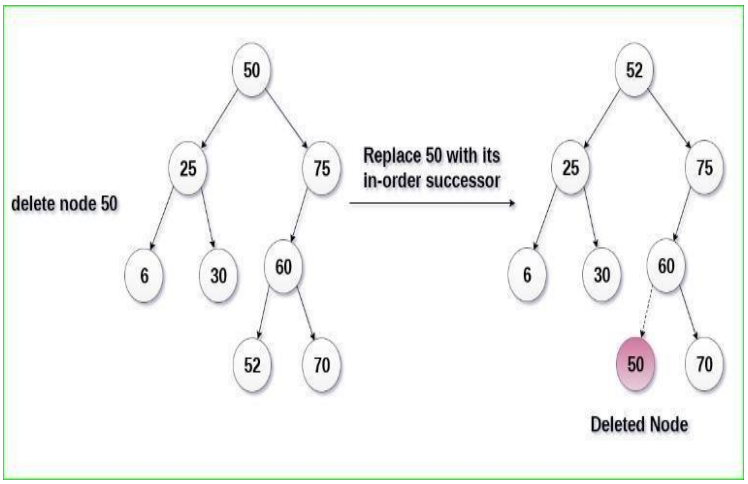
c) The node to be deleted has two children.

It is a bit complex case compared to other two cases. However, the node which is to be deleted is replaced with its in-order successor or predecessor recursively until the node value

(to be deleted) is placed on the leaf of the tree. After the procedure, replace the node with NULL and free the allocated space. In the following image, the node 50 is to be deleted which is the root node of the tree. The in-order traversal of the tree given below.

6, 25,30,50,52,60,70,75

Replace 50 with its in-order successor 52. Now, 50 will be moved to the leaf of the tree, which will simply be deleted.



7.4 AVL Tree: Construction, Traversal AVL Tree (Height Balanced Tree):

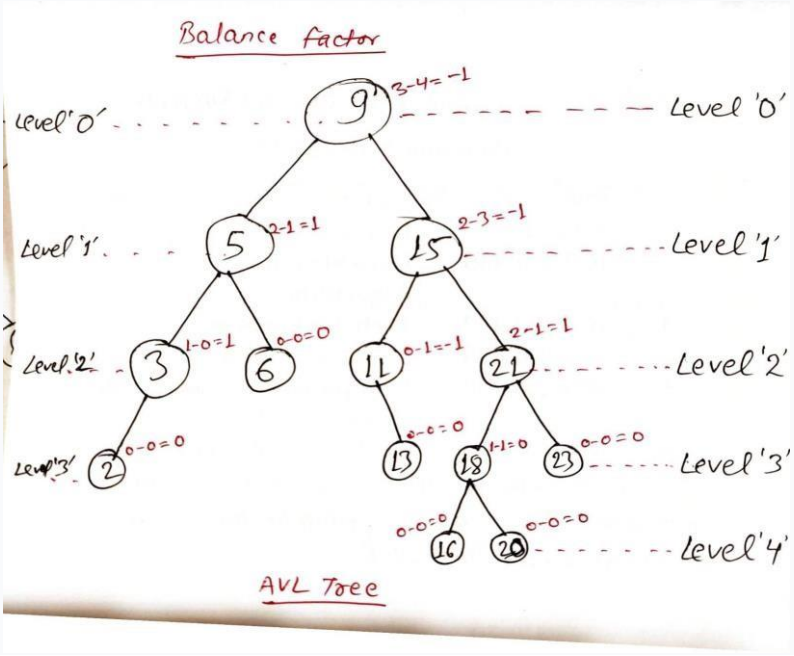
AVL tree is a self-balancing binary search tree in which each node maintains extra information called a balance factor whose value is either -1, 0 or +1. AVL tree got its name after its inventor Georgy Adelson Velsky and Landis.

Balance Factor:

Balance factor of a node in an AVL tree is the difference between the height of the left sub-tree and that of the right sub-tree of that node.

Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree). The self balancing property of an AVL tree is maintained by the balance factor. The value of balance factor should always be -1, 0 or +1.

1. The height of a tree with no elements is '0'.
2. The height of a tree with 1 element is '1'.
3. The height of a tree with >1 element is equal to 1+height of its tallest sub-tree. An example of a balanced AVL tree is:



AVL Tree Rotation:

In AVL tree, after performing every operation like insertion and deletion we need to check the balance factor of every node in the tree. If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced. We use rotation operations to make the tree balanced whenever the tree is becoming imbalanced due to any operation.