Tree

- Trees are **hierarchical** and non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels.

- ## Tree Vocabulary:
  - **Root:** The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. If a node is directly linked to some other node, it would be called a parent-child relationship.
  - **Child node:** If the node is a descendant of any node, then the node is known as a child node.
  - **Parent:** If the node contains any sub-node, then that node is said to be the parent of that sub-node.
  - **Sibling:** The nodes that have the same parent are known as siblings.
  - **Leaf Node:-** The node of the tree, which doesn't have any child node, is called a leaf node. A leaf node is the bottom-most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.
  - **Internal nodes:** A node has at least one child node known as an internal node.
  - **Ancestor node:-** An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors.
  - **Descendant:** The immediate successor of the given node is known as a descendant of a node.

- ## Advantages:
  1. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:
  2. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).
  3. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).
  4. Like Linked Lists and unlike Arrays, Trees don’t have an upper limit on number of nodes as nodes are linked using pointers.

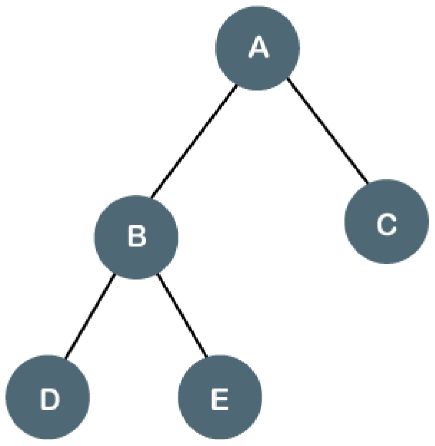
- **Applications:**
    - **Storing naturally hierarchical data:** Trees are used to store the data in the hierarchical structure. For example, the file system. The file system stored on the disc drive, the file and folder are in the form of the naturally hierarchical data and stored in the form of trees.
    - **Organize data:** It is used to organize data for efficient insertion, deletion and searching. For example, a binary tree has a logN time for searching an element.
    - **Trie:** It is a special kind of tree that is used to store the dictionary. It is a fast and efficient way for dynamic spell checking.
    - **Heap:** It is also a tree data structure implemented using arrays. It is used to implement priority queues.
    - **B-Tree and B+Tree:** B-Tree and B+Tree are the tree data structures used to implement indexing in databases.
    - **Routing table:** The tree data structure is also used to store the data in routing tables in the routers.

Binary Tree

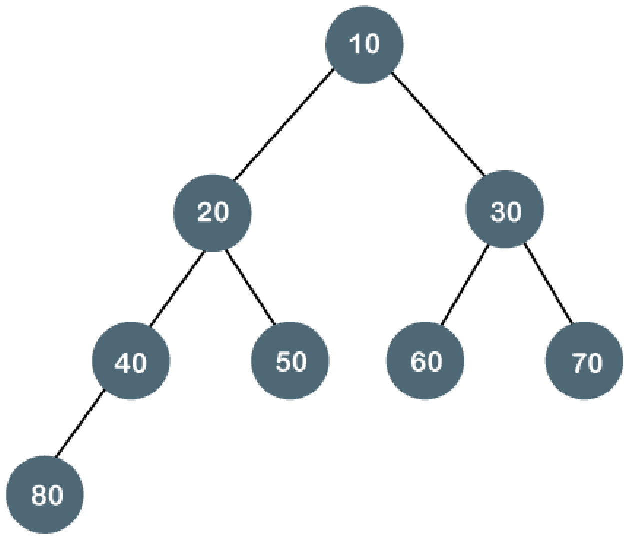
- **Binary Tree:** A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

- **Type of Binary Tree**

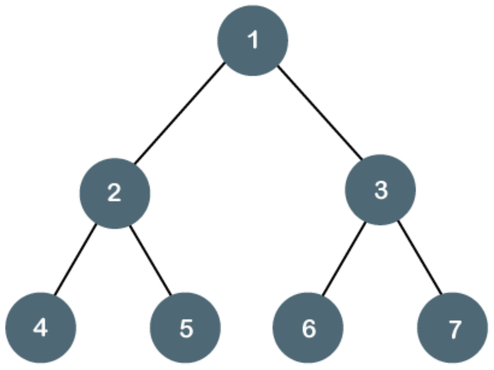
1. **Full / Proper / Strict Binary Tree:** The full binary tree is also known as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.



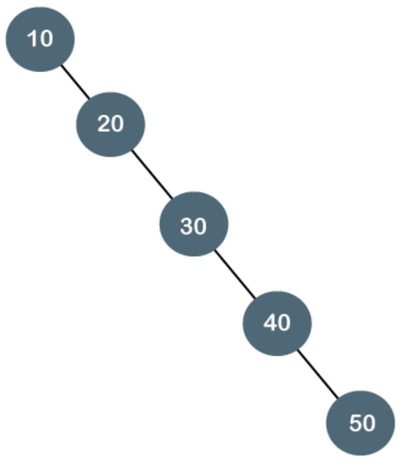
2. **Complete Binary Tree:** The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.



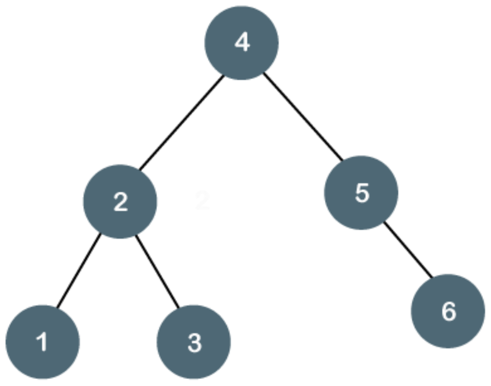
3. **Perfect Binary Tree:** A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.



4. **Degenerate Binary Tree:** The degenerate binary tree is a tree in which all the internal nodes have only one child.



5. **Balanced Binary Tree:** The balanced binary tree is a tree in which both the left and right trees differ by atmost 1. For example, ***AVL*** and ***Red-Black trees*** are balanced binary tree.



- **Binary Tree Representation in C:** A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL.
  1. A Tree node contains following parts.
     - Data
     - Pointer to left child
     - Pointer to right child

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

- **Properties of Binary Tree:**

1. **The minimum number of nodes at height h:**
   In any binary tree, the minimum number of nodes will be one more than the value of height. This is the number of nodes required to construct a tree.

   ***Formula**:*
   Height = h.
   The minimum number of nodes = h + 1.
   If h = 3, then nodes will be 3 + 1 = 4.

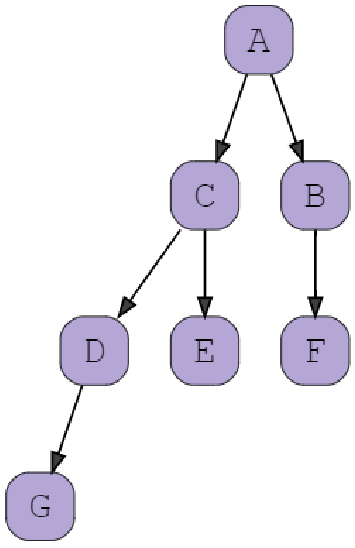
2. **The maximum number of nodes at height h**:
   The maximum number of nodes that can be inserted in any binary tree of height **h** will be equal to 2^h- 1.

   ***Formula**:*
   Height = h.
   Maximum nodes to inserted = 2^h - 1.
   If h = 3, then 2^3 - 1.
   8 - 1 = 7.
   Therefore, the maximum number of nodes to be inserted for the height of h = 3 will be **7**.

3. **Total number of leaf nodes:**
   The number of leaf nodes in a binary tree is equal to the nodes with degree two, plus one. Say a binary tree has two children. Then the total number of leaf nodes of that binary tree will be one greater than the nodes having two children.

   ***Formula**:*
   Total number of leaf nodes = Nodes with 2 children + 1

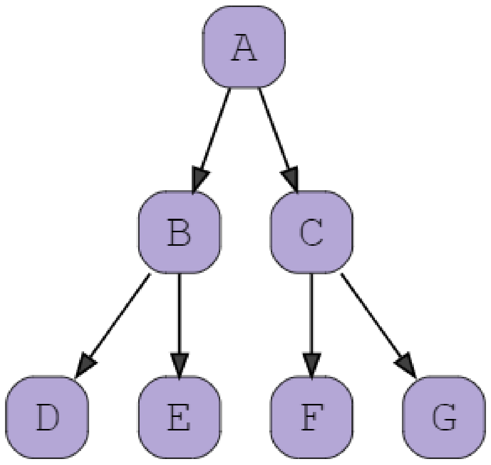
Example:
Here, the total number of leaf nodes (no child or a successor) is 3, and the leaf nodes are E, F, and G.
Whereas the nodes with two children are 2. Node A and C have 2 children. Hence, this proves the property.



4. **The maximum number of nodes at any level:**
   In a binary tree, the maximum number of nodes gained by any level is equal to the power of 2 for that level.

   In a simpler word, the level is donated by n. Then, maximum nodes of that binary tree will be 2^n.

   Example:
   n = 2 then $2^2 = 4$.
   Level 2 coverts the nodes (D, E, F, and G).



5. **Minimum possible height or levels is equal to $Log_2(N+1)$:**
   This property says that the minimum number of levels or a height of a binary tree is the Log_2 of (N+1). Here N represents the maximum number of nodes possessed by a binary tree at the height h.

   We have already discussed property 2 along with the example. Let's use that answer to calculate the height h.

   N = 7 (Using Property#2)

   Log_2 of (N+1) = Log_2 of (7+1)

   $Log_2(8) = 3$

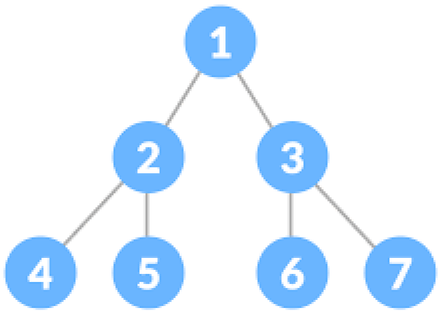
   It is equal to the height chosen in the example of property 2.

- ## Tree Traversal

  A Tree is typically traversed in two ways:

  1. Breadth First Traversal (BFT) Or Level Order Traversal
  2. Depth First Traversals (DFT)
     - In Order Traversal (Left-Root-Right)
     - Pre Order Traversal (Root-Left-Right)
     - Post Order Traversal (Left-Right-Root)

  Example:



• Level Order Traversal	: 1 2 3 4 5 6 7
• Pre Order Traversal	: 1 2 4 5 3 6 7
• In Order Traversal	: 4 2 5 1 3 6 7
• Post Order Traversal	: 4 5 2 6 7 3 1

Binary Search Tree (BST)

- A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.

- **Advantages:**
    - Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.
    - As compared to array and linked lists, insertion and deletion operations are faster in BST.

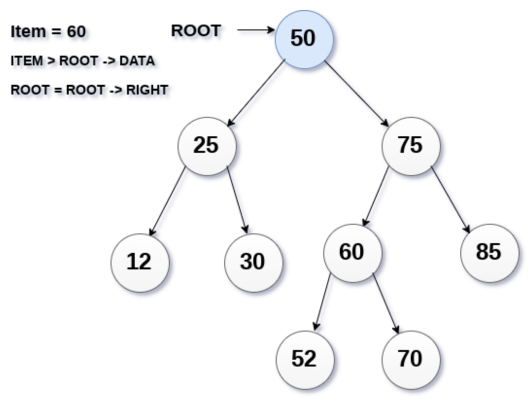
- **Time Complexity:**

Operations	Best case time complexity	Average case time complexity	Worst case time complexity
Insertion	O(log n)	O(log n)	O(n)
Deletion	O(log n)	O(log n)	O(n)
Search	O(log n)	O(log n)	O(n)

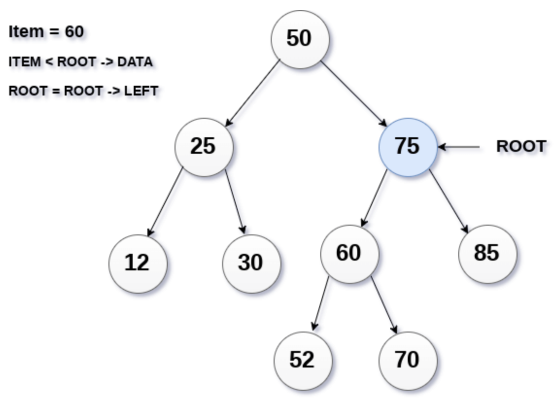
- **Space Complexity:**

Operations	Space complexity
Insertion	O(n)
Deletion	O(n)
Search	O(n)

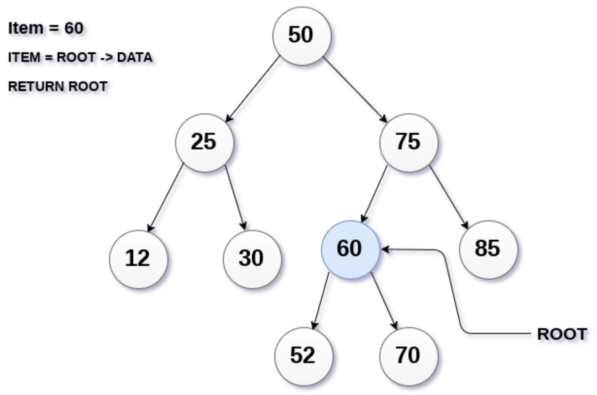
- **Searching:**
  - Searching means finding or locating some specific element or node within a data structure. However, searching for some specific node in binary search tree is pretty easy due to the fact that, element in BST are stored in a particular order.
  - **Algo:**
    - Compare the element with the root of the tree.
    - If the item is matched, then return the location of the node.
    - Otherwise check if item is less than the element present on root, if so then move to the left sub-tree.
    - If not, then move to the right sub-tree.
    - Repeat this procedure recursively until match found.
    - If element is not found, then return NULL.



STEP 1



STEP 2

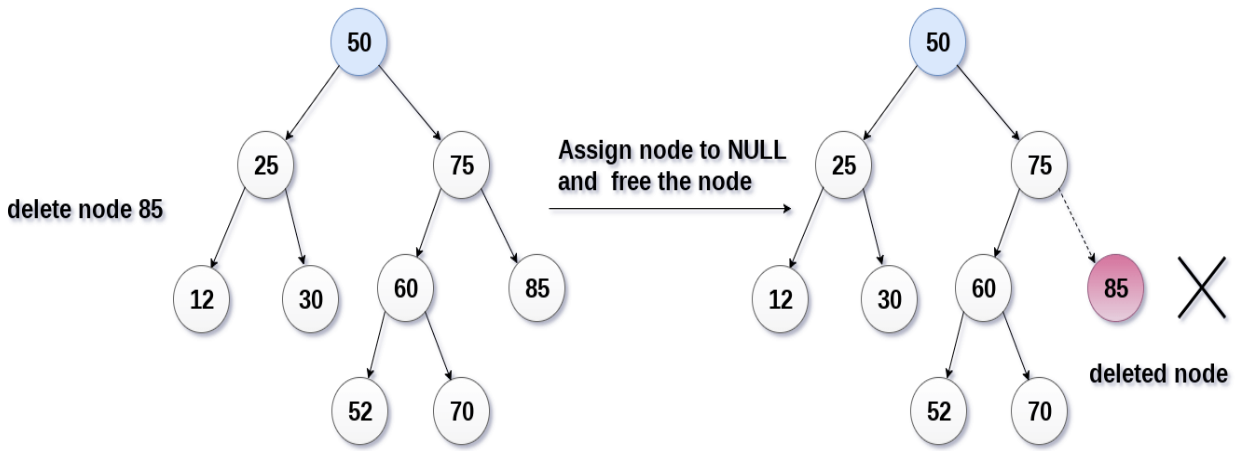


STEP 3

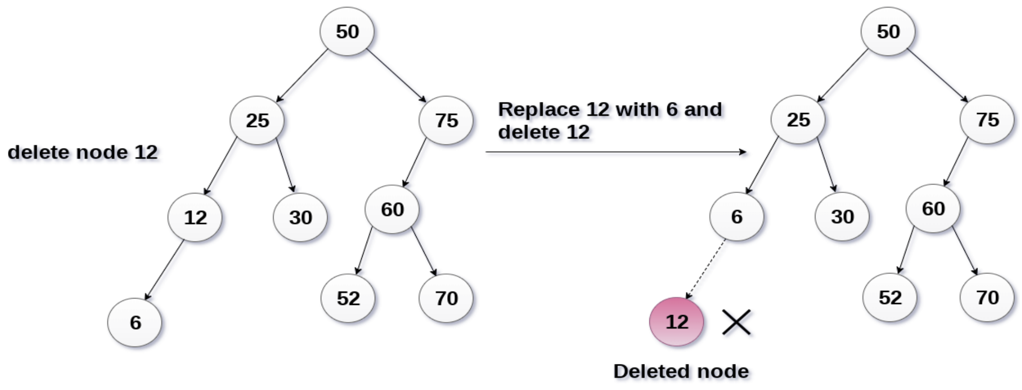
- **Insertion:**
  - Insert function is used to add a new element in a binary search tree at appropriate location. Insert function is to be designed in such a way that, it must node violate the property of binary search tree at each value.
  - **Algo:**
    - Allocate the memory for tree.
    - Set the data part to the value and set the left and right pointer of tree, point to NULL.
    - If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
    - Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.
    - If this is false, then perform this operation recursively with the right sub-tree of the root.

- **Deletion:**
  - Delete function is used to delete the specified node from a binary search tree. However, we must delete a node from a binary search tree in such a way, that the property of binary search tree doesn't violate.
  - **Algo:** There are three situations of deleting a node from binary search tree.

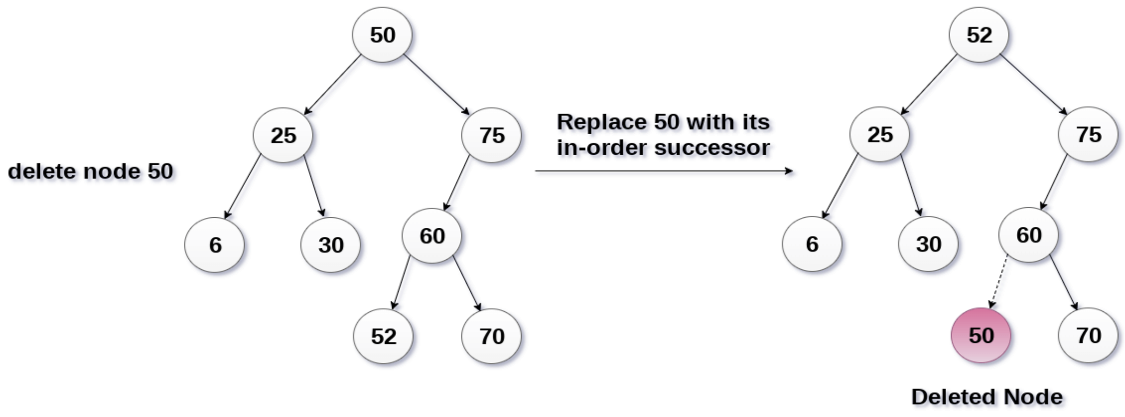
1. **The node to be deleted is a leaf node:**
   - It is the simplest case, in this case, replace the leaf node with the NULL and simple free the allocated space.
   - In the following image, we are deleting the node 85, since the node is a leaf node, therefore the node will be replaced with NULL and allocated space will be freed.



2. **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.



3. **The node to be deleted has two children:**
   - It is a bit complexed case compare 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.


delete node 50
50
25
75
6
30
60
52
70
Replace 50 with its in-order successor
52
25
75
6
30
60
50
70
Deleted Node