Tree Data Structure

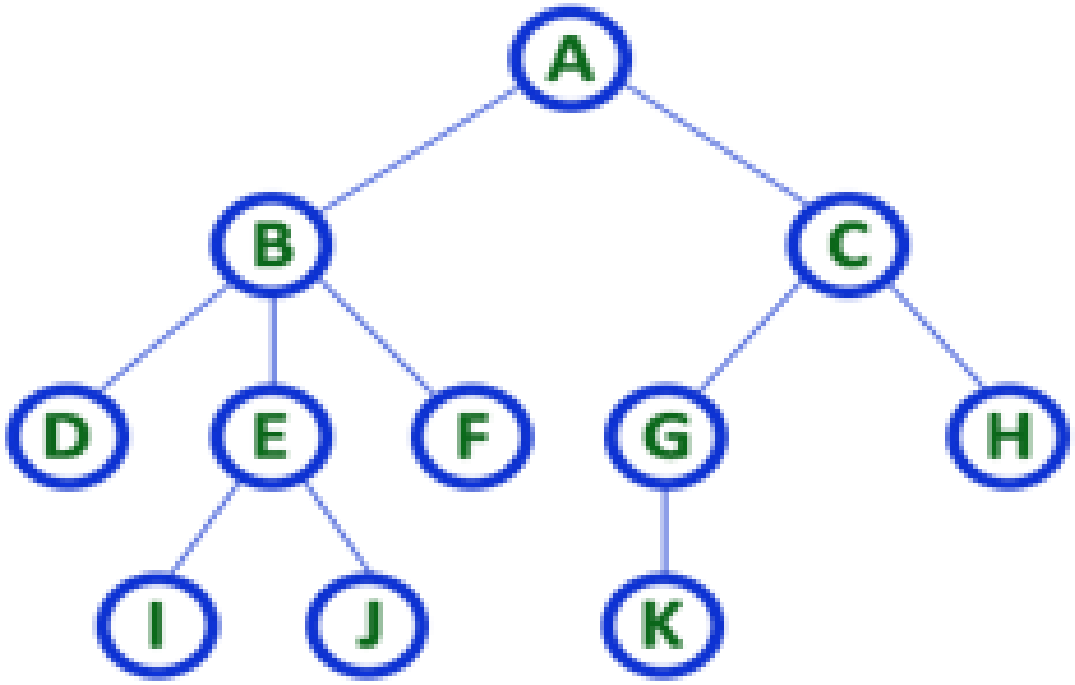
Dr.M.RAJAMANI

Asst. Professor

CSE, GST

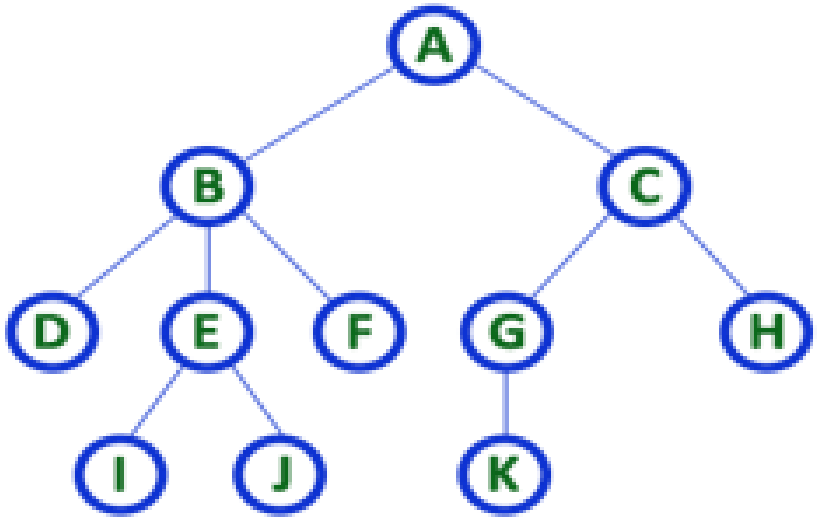
TREE Data Structure

- A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.



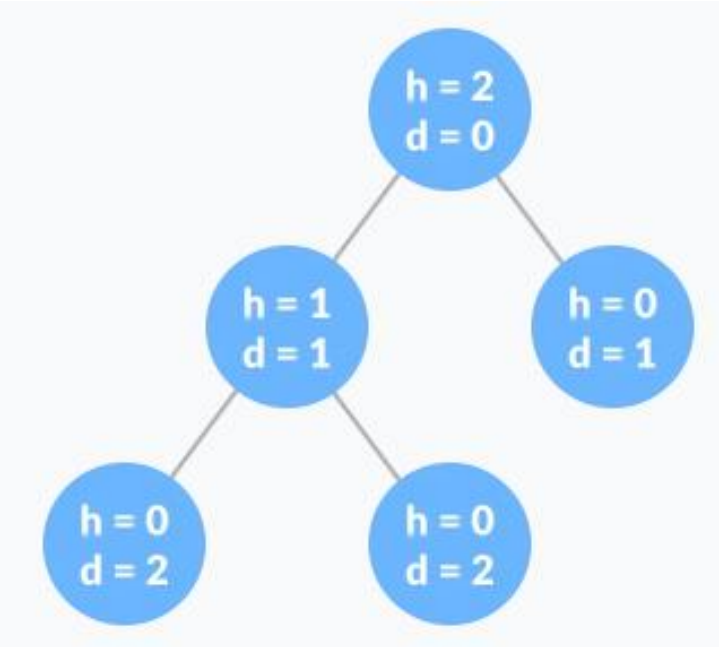
Tree Terminology

- **Node :** A node is an entity that contains a key or value and pointers to its child nodes. Eg: A,B,C,D,E,F,G,H,I,J,K
- Leaf / External Nodes : The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes. Eg: D,F,H,I,J,K
- Internal Nodes : The node having at least a child node is called an **internal node**. Eg: A,B,C,E,G
- Edge : It is the link between any two nodes.
- Root : It is the topmost node of a tree.



Tree Terminology

- Height of a node : The number of edges from the node to the deepest leaf (i.e. the longest path from the node to a leaf node).
- Depth of a node : The number of edges from the root to the node.
- Height of a tree : The total number of levels in a tree is the height of a tree. It is also known as depth of the tree.
- Degree of a node : The total number of children of a node.
- Degree of Tree : The highest degree of the node among all the nodes in a tree.



Tree Terminology

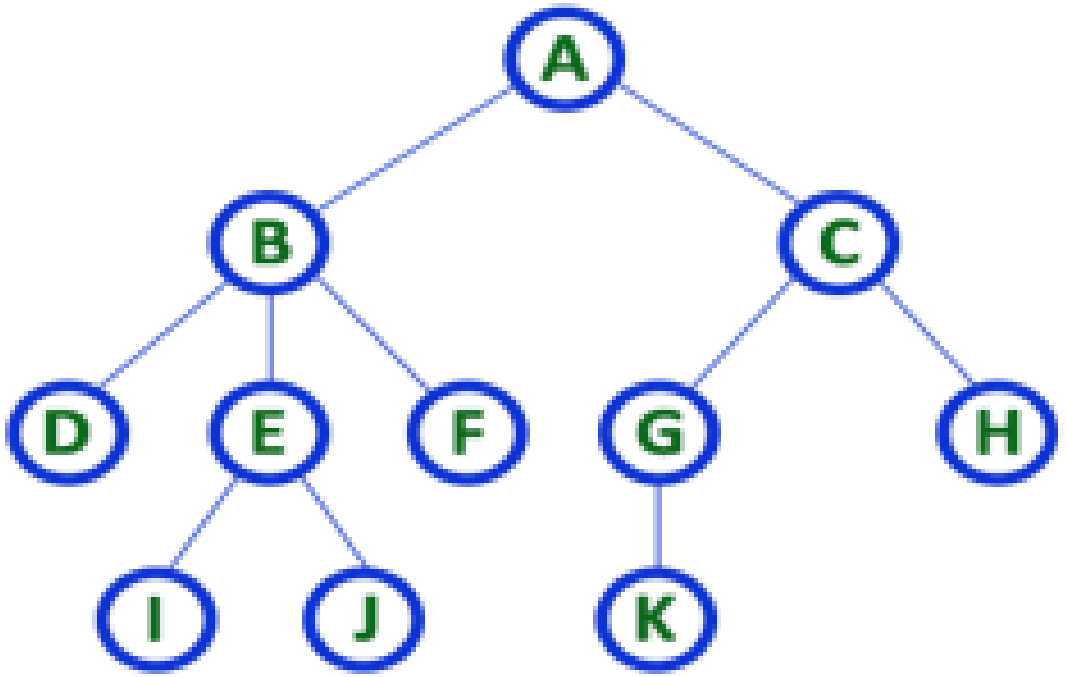
- An ancestor of a node is the parent of a node or the nodes in linage to the root node.
- The descendant of a node is a child of a node or the nodes in linage to the leaf node.

Types of Trees

1. General Tree
2. Binary Tree
3. Binary Search Tree
4. AVL Tree
5. B Tree

General Tree

- A Tree in which each node can have 0 or more children.



Binary Tree

- *A binary tree is a tree data structure in which each node can have at most two children*
  - *left child and the right child.*
- *Applications*
  - Data storage and retrieval
  - Expression evaluation
  - Network routing
  - Implementation of various searching, sorting, and graph algorithms.

Properties of a Binary Tree

1. The maximum number of nodes at level ‘l’ of a binary tree is 2^l. The level of the root is 0.
2. The minimum number of nodes in a binary tree of height ‘h’ is h.
3. The maximum number of nodes in a binary tree of height ‘h’ is $2^h – 1$.
4. In a binary tree with N nodes, the maximum possible height is N (Skew tree), the minimum possible height or the minimum number of levels is $Log_2(N+1)$.
5. In a binary tree where every node has 0 or 2 children, the number of leaf nodes is always one more than nodes with two children.

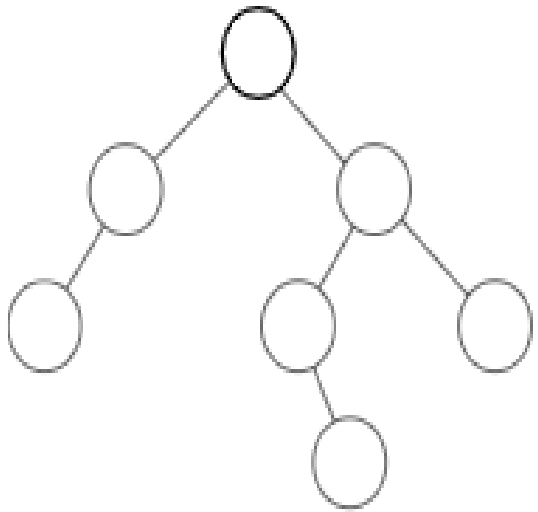
   (If N_0 is the no. of leaf nodes and N_2 is the no. of nodes with 2 children then $N_0=N_2+1$)

6. In a non-empty binary tree, if ‘n’ is the total number of nodes and ‘e’ is the total number of edges, then e = n-1.

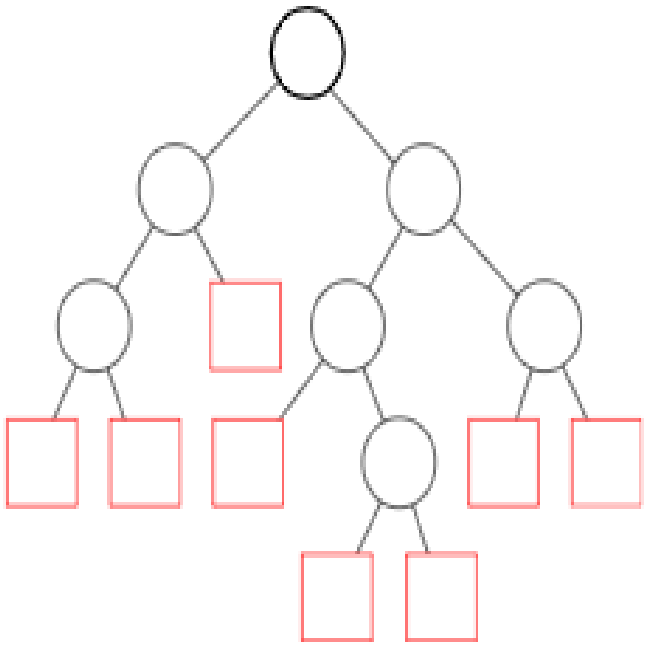
Types of Binary Trees

1. **Extended Binary Tree**
2. **Complete Binary Tree**
3. **Full Binary Tree**
4. **Skewed Binary Tree**
5. **Strictly Binary Tree**

1. Extended binary tree is a type of binary tree in which all the null sub tree of the original tree are replaced with special nodes called **external nodes** whereas other nodes are called **internal nodes**

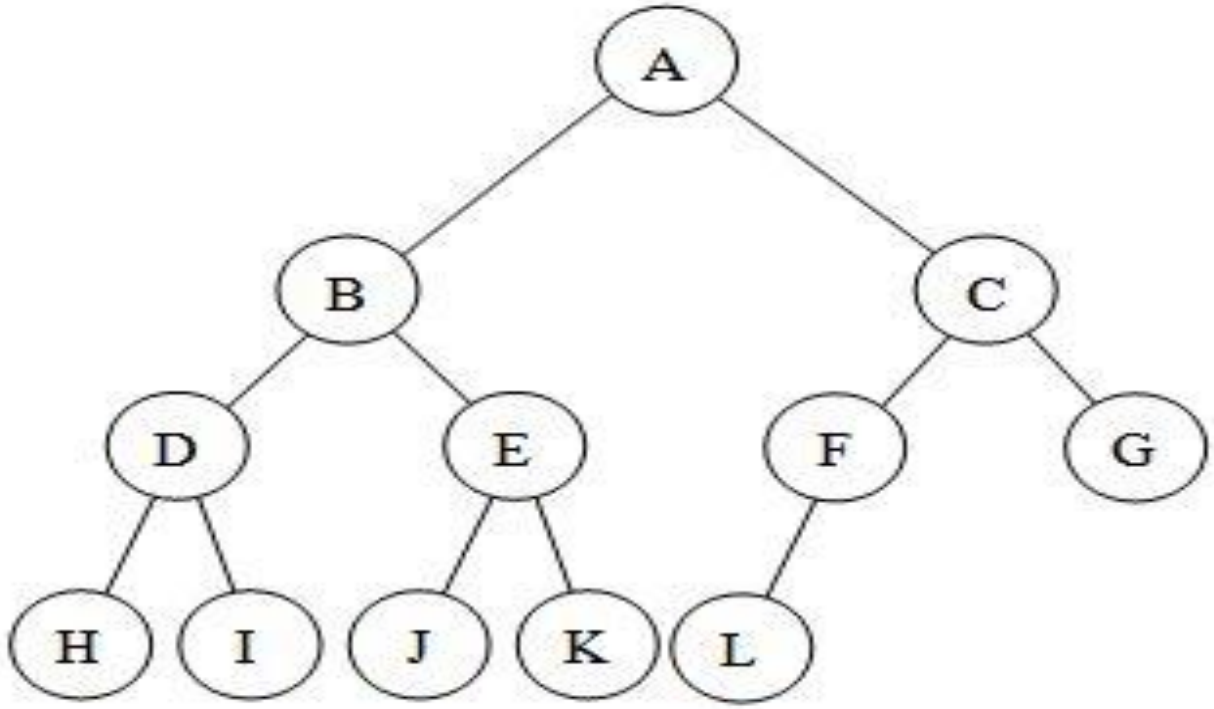


Binary Tree



Exended Binary Tree

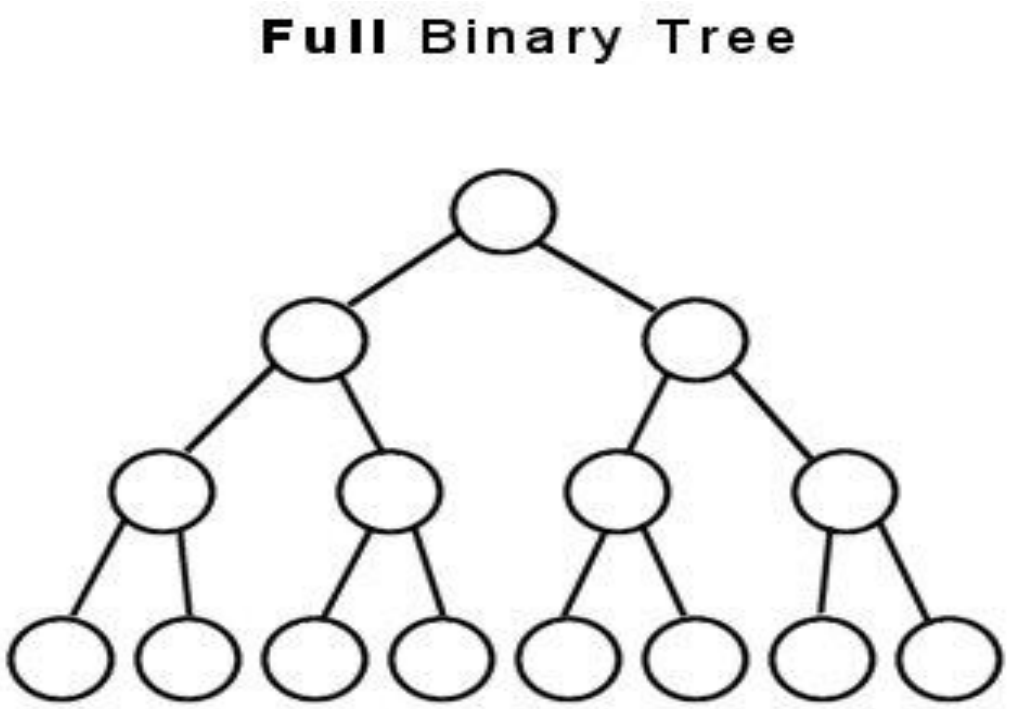
2. Complete Binary Tree has all levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.



3. A binary tree is a **full binary tree** if all levels have maximum number of nodes.

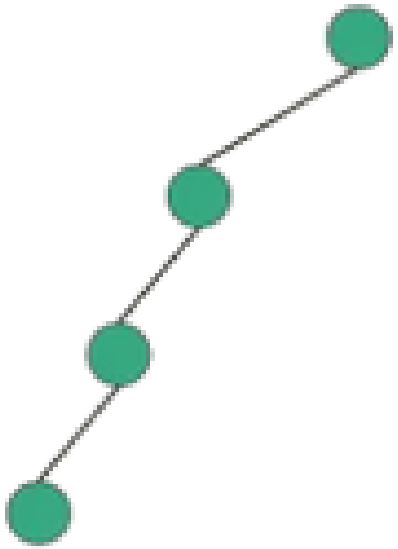
Note :

- If ‘h’ is the height of the tree then the maximum number of nodes is $2^h - 1$
- Every level is completely filled up.

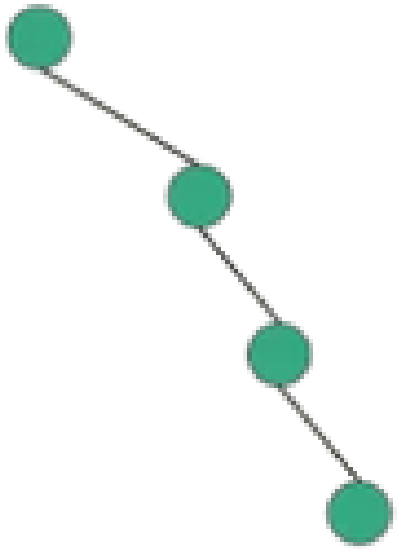


4. Skew Tree is a tree in which it is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: **left-skewed binary tree** and **right-skewed binary tree**.

Note : The trees which have minimum number of nodes

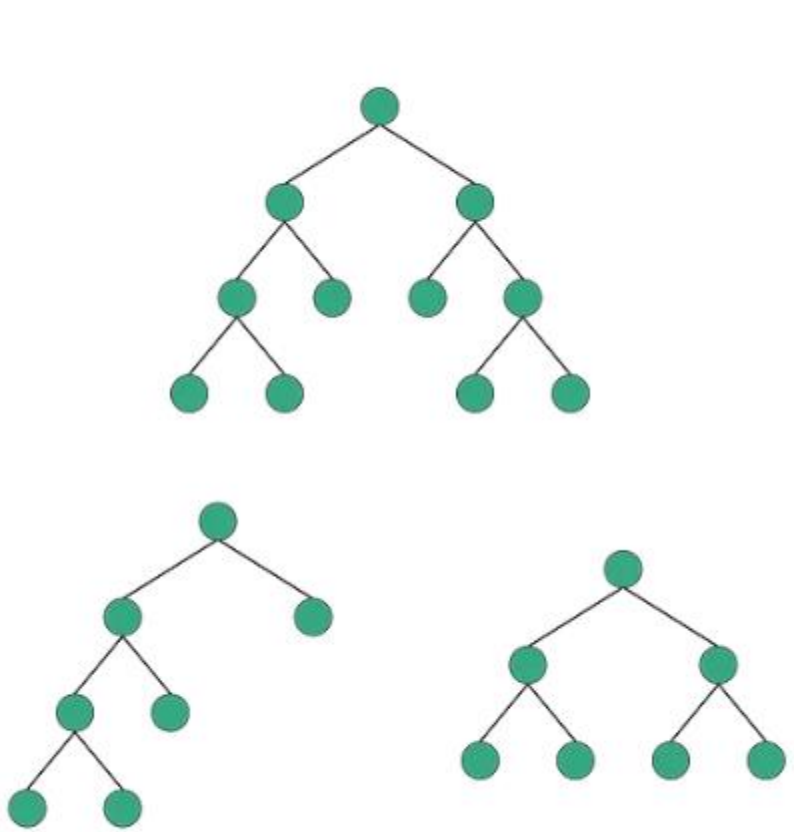


Left Skewed Binary Tree

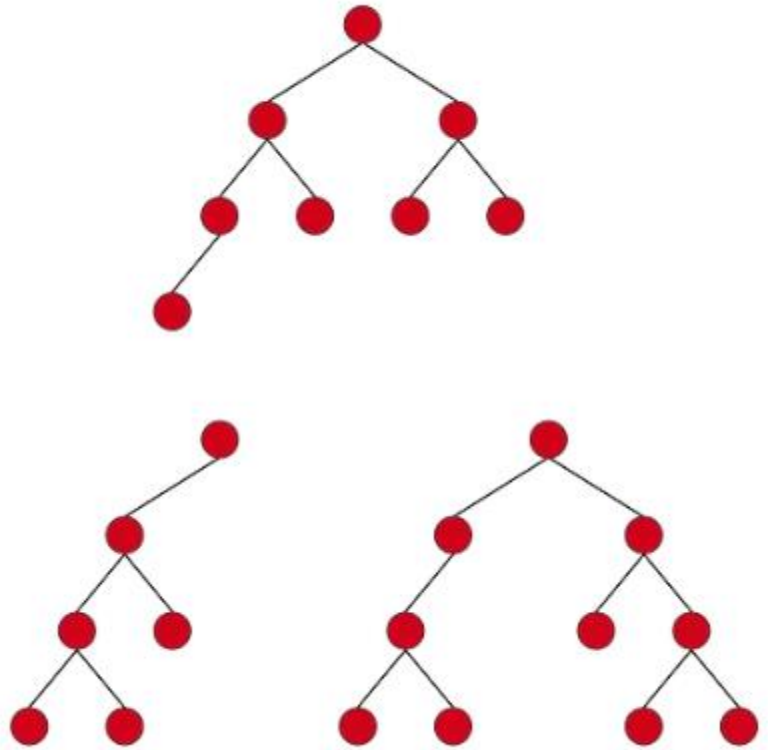


Right Skewed Binary Tree

5. Strictly Binary Tree is a Binary Tree in which every node has 0 or 2 children.



Valid Strictly Binary Trees



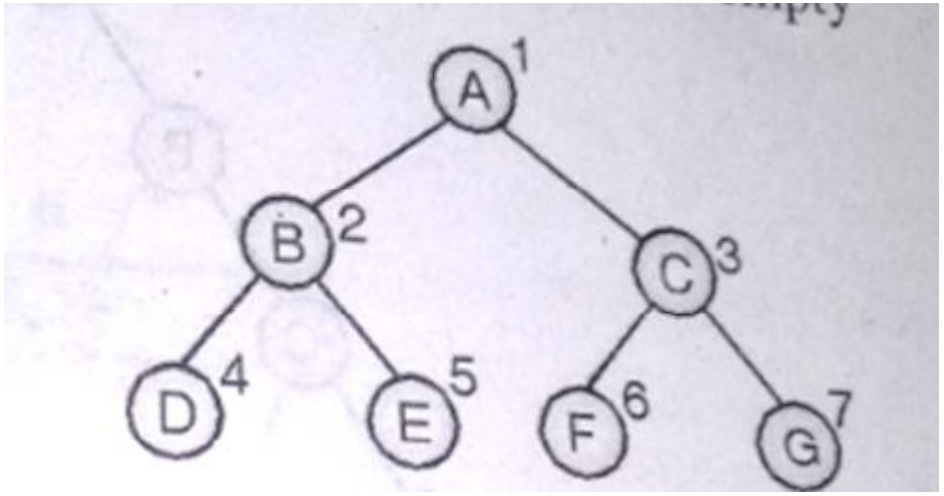
Invalid Strictly Binary Trees

Representation of Binary Trees

1. Array Representation
2. Linked List Representation

Array Representation

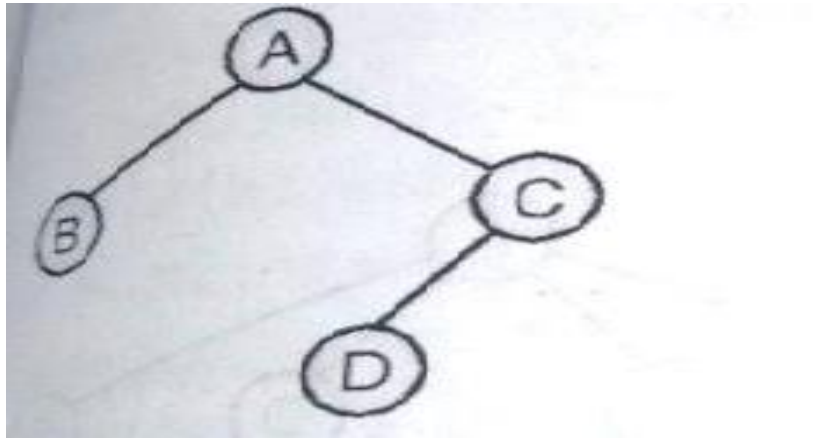
1. To represent a tree in one dimensional array nodes are marked sequentially from left to right start with root node.
2. First array location can be used to store no of nodes in a tree.

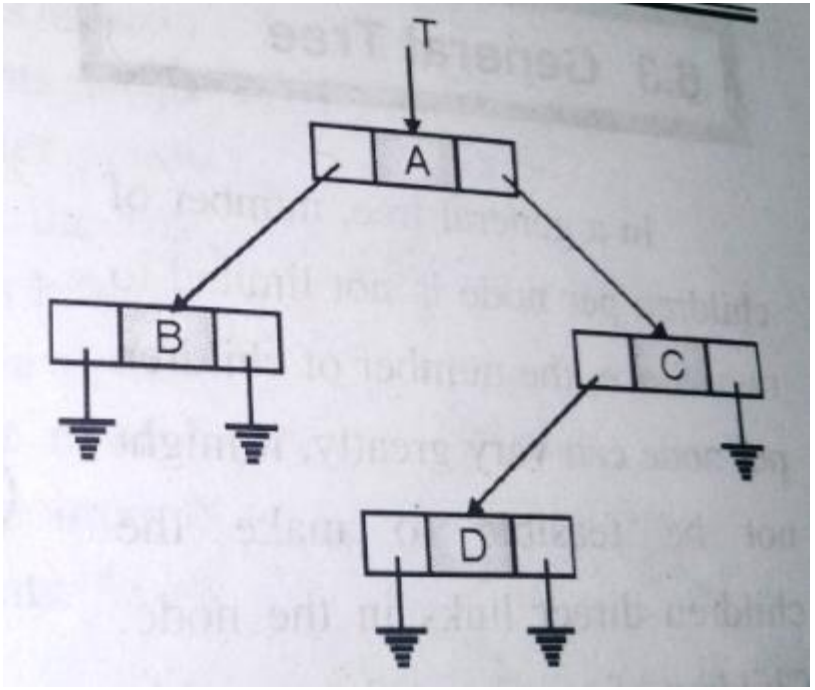


0	1	2	3	4	5	6	7
7	A	B	C	D	E	F	G

Linked List Representation

1. This type of representation is more efficient as compared to array.
2. Left and right are pointer type fields left holds address of left child and right holds address of right child.
3. ***struct node***
   { int data;
   struct node * left,*right;
   };





Traversal in Binary Trees

- Traversing a binary tree means visiting each node of the tree exactly once. Traversal of tree gives linear order of the nodes.
- Preorder Traversal
- Inorder Traversal
- Postorder Traversal

Inorder Traversal

```
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}
```

Preorder Traversal

```
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
```

Post order Traversal

```
void postorderTraversal(struct node* root) {
  if (root == NULL) return;
  postorderTraversal(root->left);
  postorderTraversal(root->right);
  printf("%d ->", root->item);
}
```

BINARY SEARCH TREES

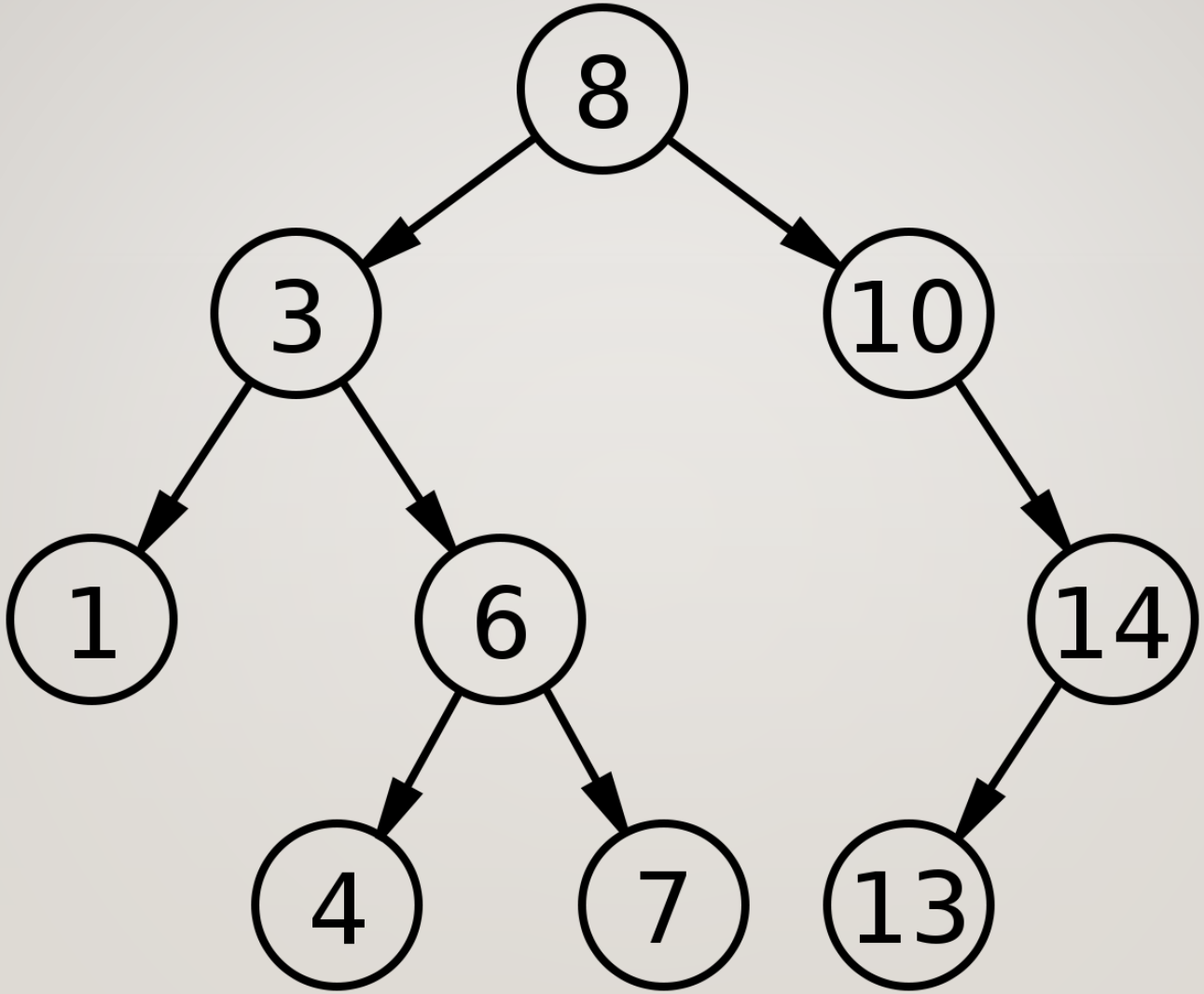
BINARY SEARCH TREE

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

- It is called a binary tree because each tree node has a maximum of two children.
- It is called a search tree because it can be used to search for the presence of a number in $O(\log(n))$ time.

The properties that separate a binary search tree from a regular binary tree is

1. All nodes of left subtree are less than the root node
2. All nodes of right subtree are more than the root node
3. Both subtrees of each node are also BSTs i.e. they have the above two properties


8
3
10
1
6
14
4
7
13

SEARCH

```
If root == NULL
     return NULL;
If number == root->data
     return root->data;
If number < root->data
     return search(root->left)
If number > root->data
     return search(root->right)
```

INSERT A NODE INTO A BINARY TREE

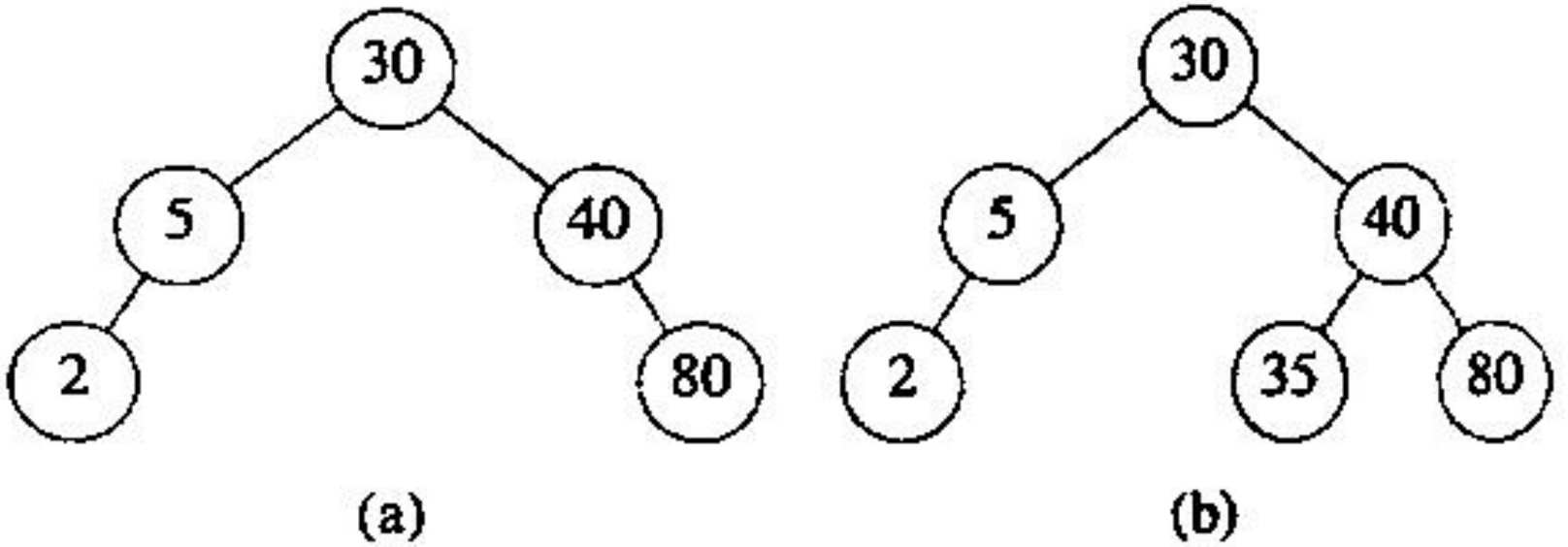


Figure 5.29: Inserting into a binary search tree

INSERT

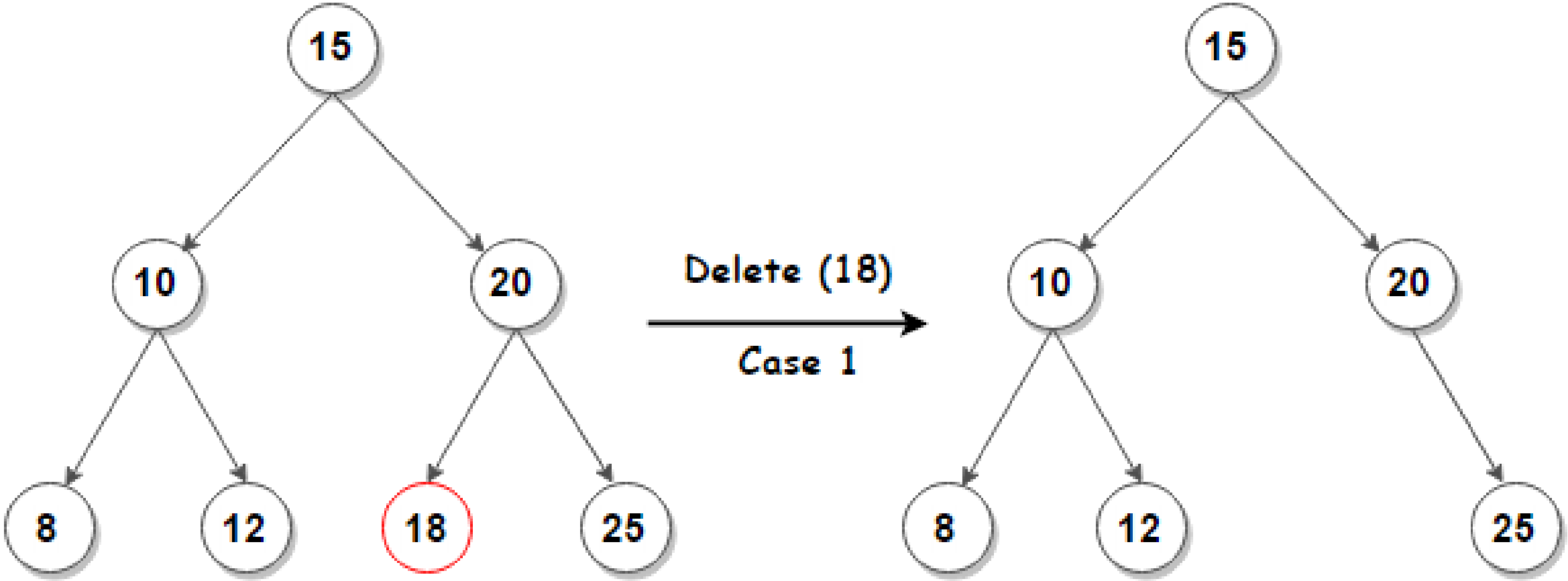
```
If node == NULL
    return createNode(data)
if (data < node->data)
    node->left  = insert(node->left, data);
else if (data > node->data)
    node->right = insert(node->right, data);
return node;
```

DELETE A NODE FROM BST

- There are three cases for deleting a node from a binary search tree.
- **Case I : T**he node to be deleted is the leaf node. In such a case, simply delete the node from the tree.
- **Case II : T**he node to be deleted lies has a single child node. In such a case follow the steps below:
    1. Replace that node with its child node.
    2. Remove the child node from its original position.
- **Case III : T**he node to be deleted has two children. In such a case follow the steps below:
    1. Get the inorder successor of that node.
    2. Replace the node with the inorder successor.
    3. Remove the inorder successor from its original position.

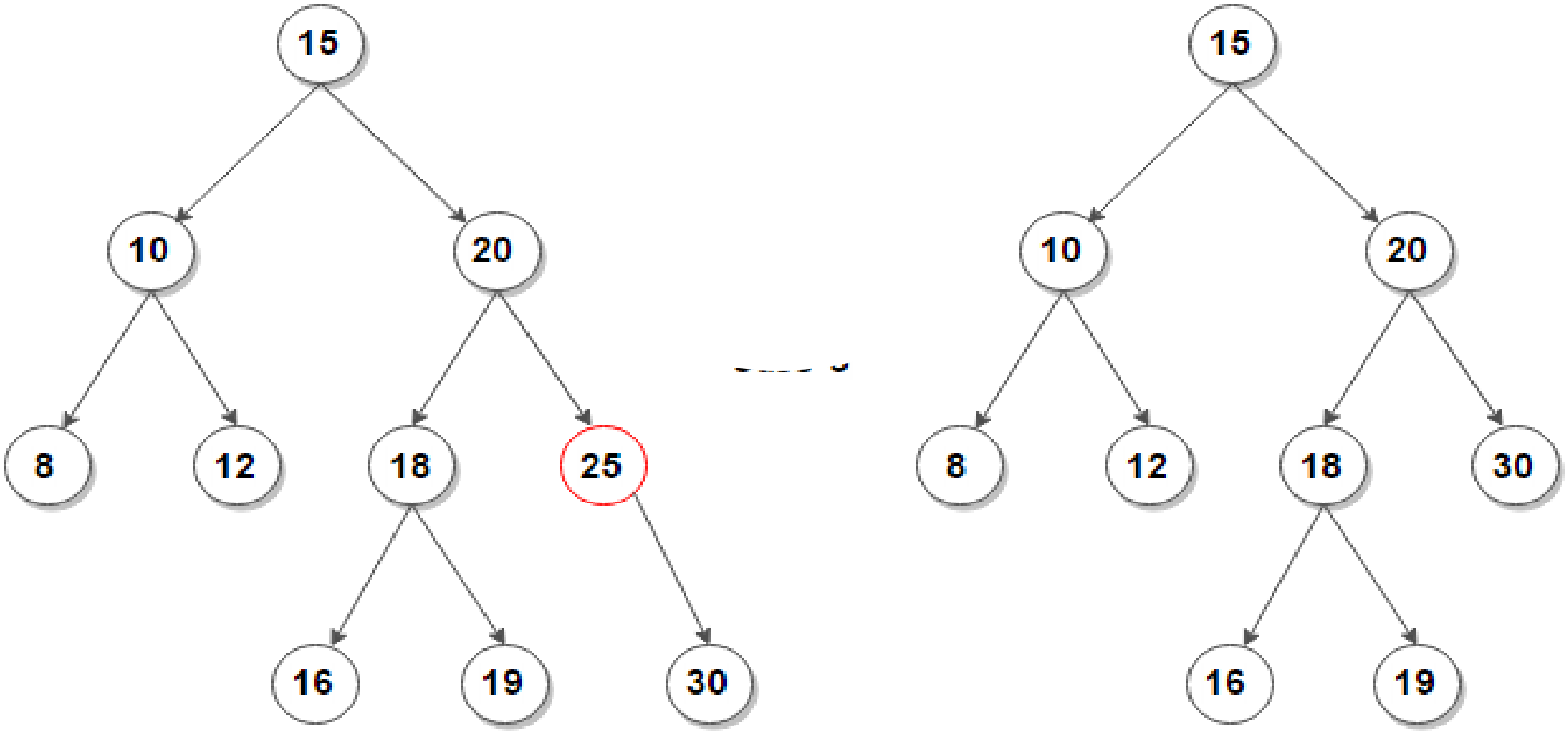
CASE I

Deleting a node with no children: remove the node from the tree.

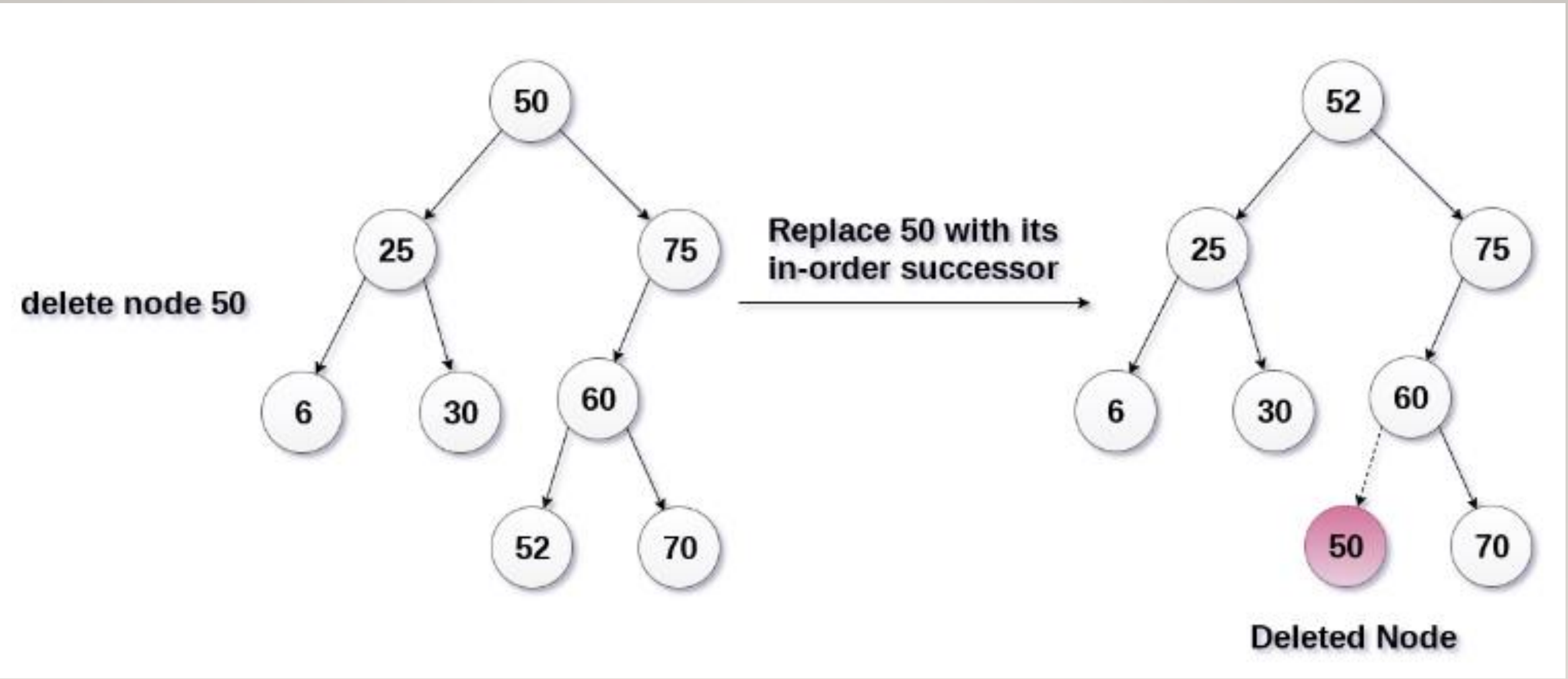


CASE II

Deleting a node with one child: remove the node and replace it with its child.



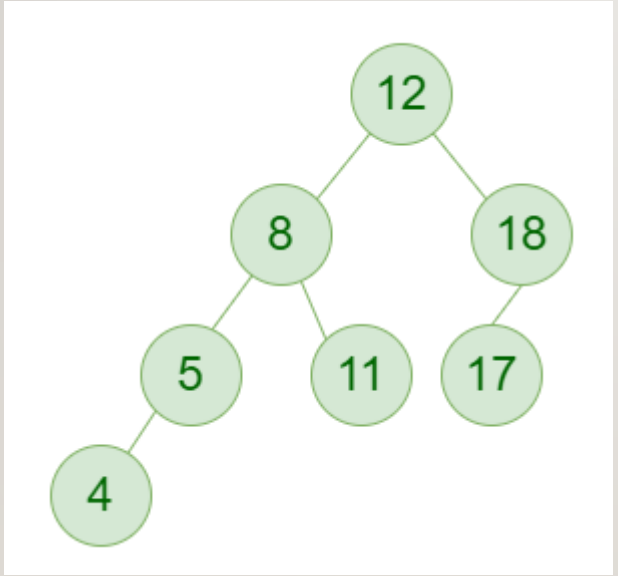
CASE III : DELETING A NODE WITH TWO CHILDREN

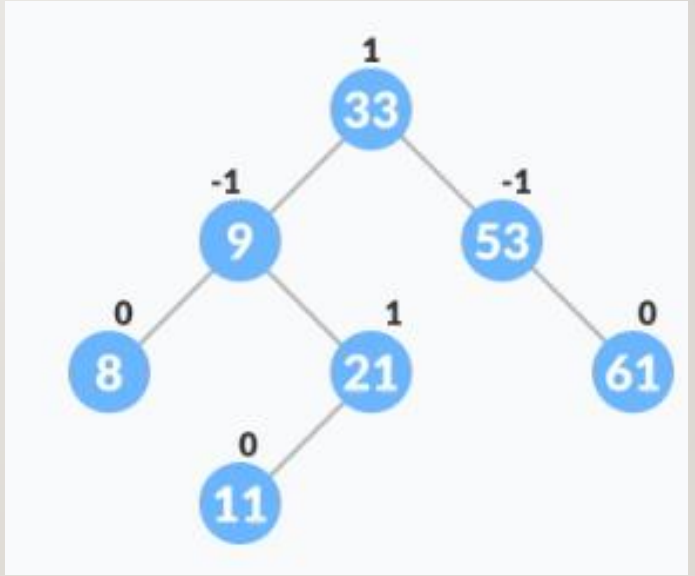


AVL TREES

AVL TREE

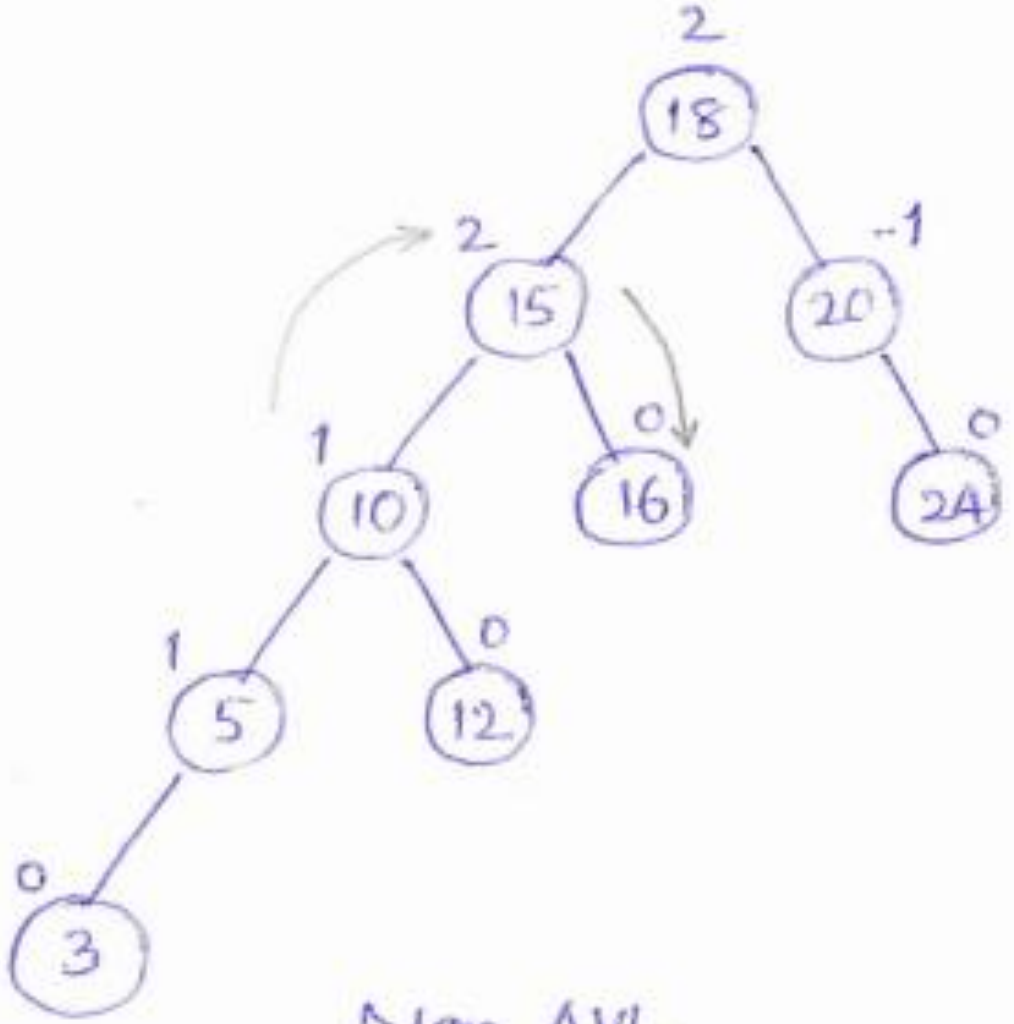
- *An **AVL tree** is defined as a **height-balanced / self-balancing** Binary Search Tree(BST) where the balance factor is* either -1, 0 or +1.
- Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree)
- AVL tree got its name after its inventor Georgy Adelson-Velsky and Landis.



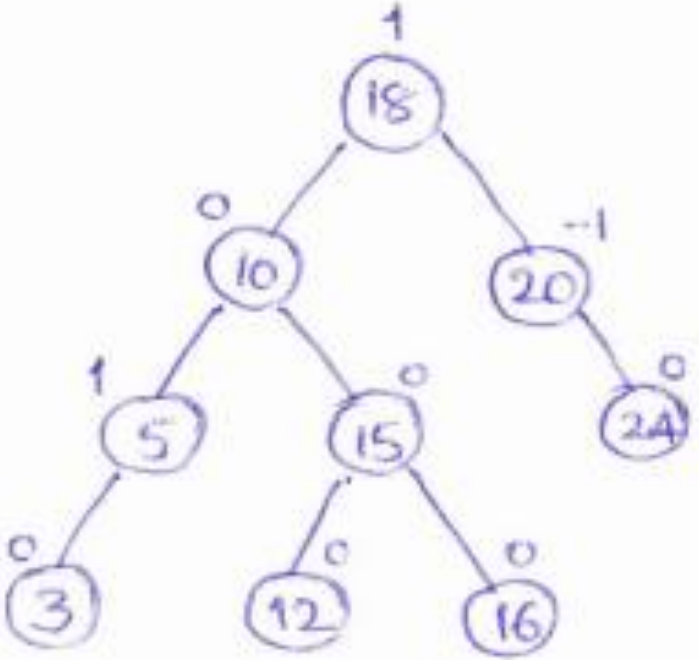


BALANCE FACTOR :

- If Balance Factor > 0 : left-heavy because the height of the left subtree is greater than the height of the right subtree
- If Balance factor < 0 : right-heavy because the height of the left subtree is less than the height of the right subtree
- If Balance factor ==0 : the subtree is perfectly balanced, with equal heights in both the left and right subtrees.
- **If any node of the tree falls out of balance after insertion or deletion, the necessary rotations are to be carried out to correct the imbalance.**



Non AVL



AVL TREE