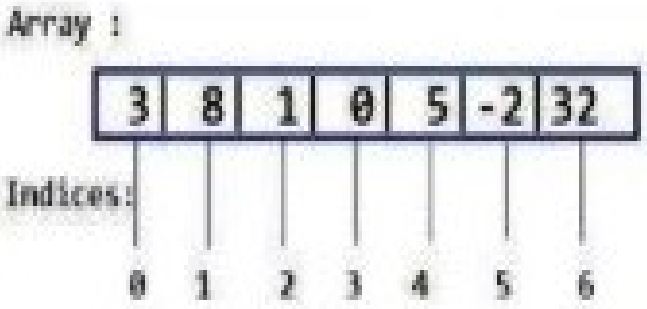
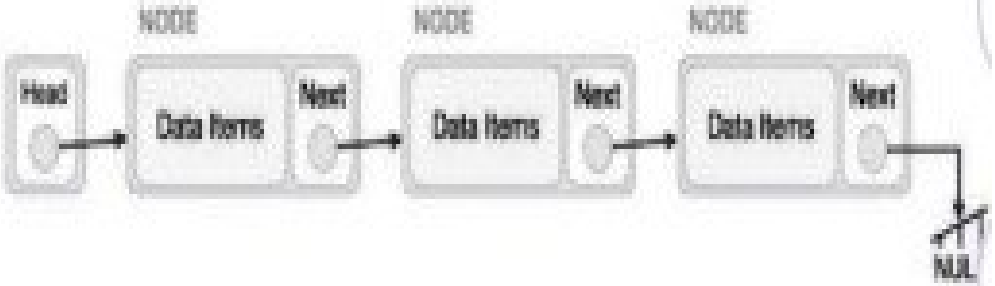
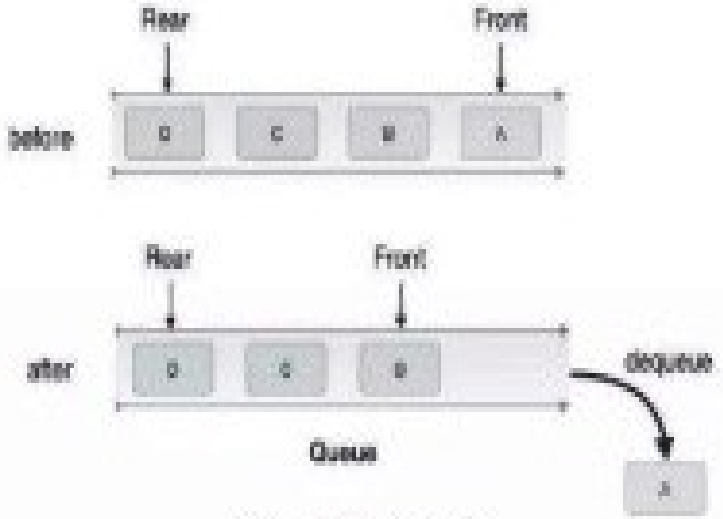
Binary Tree

So far we discussed Linear data structures like





stack



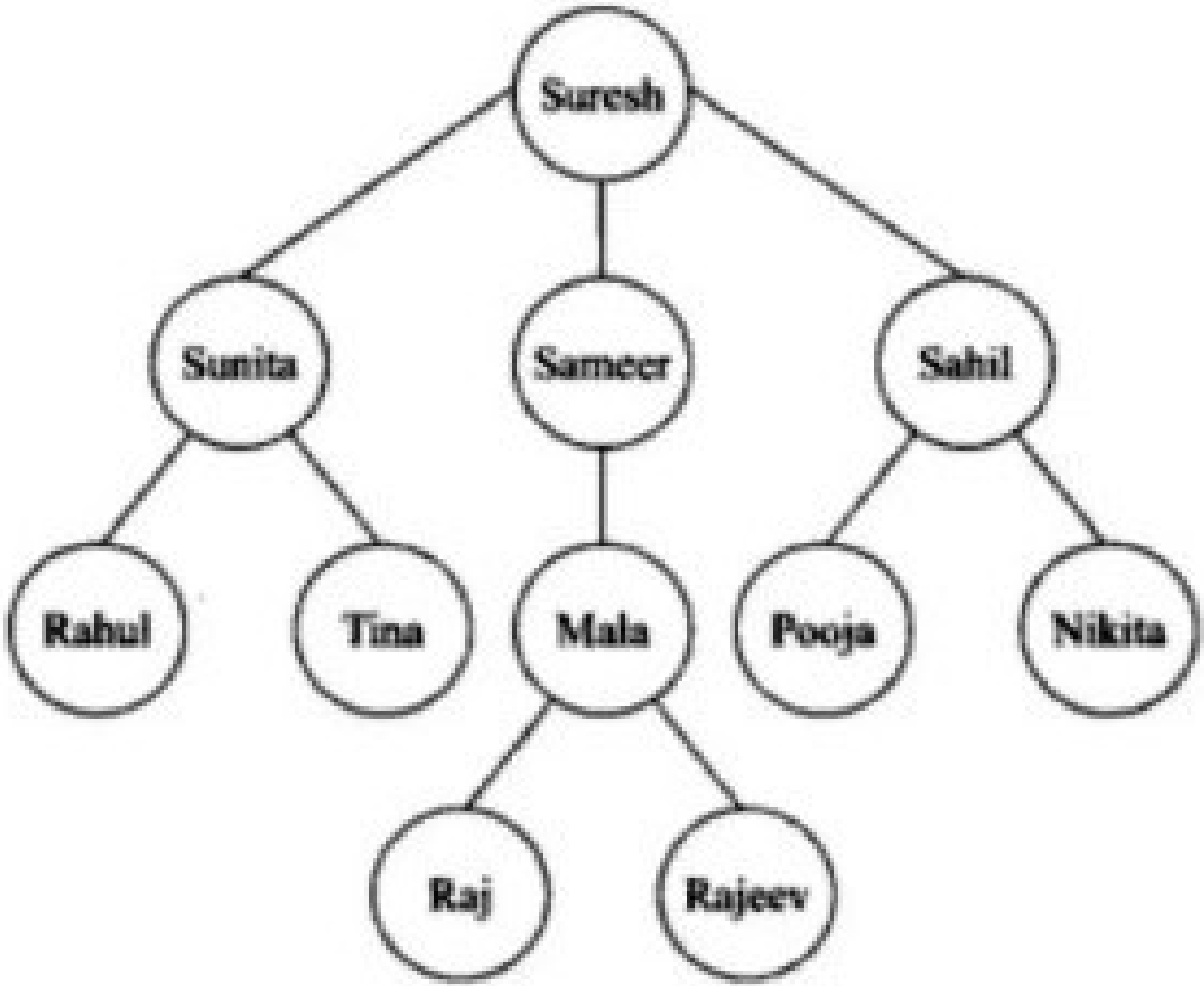
Queue Dequeue

Introduction to trees

- So far we have discussed mainly linear data structures – strings, arrays lists, stacks and queues

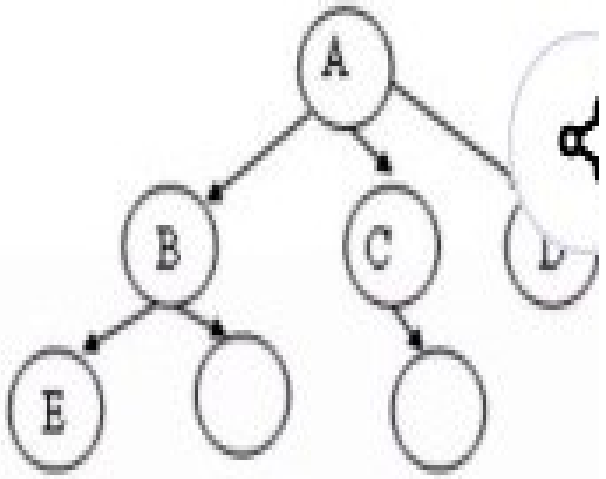
- Now we will discuss a non-linear data structure called **tree**.

- Trees are mainly used to represent data containing a hierarchical relationship between elements, for example, records, family trees and table of contents.

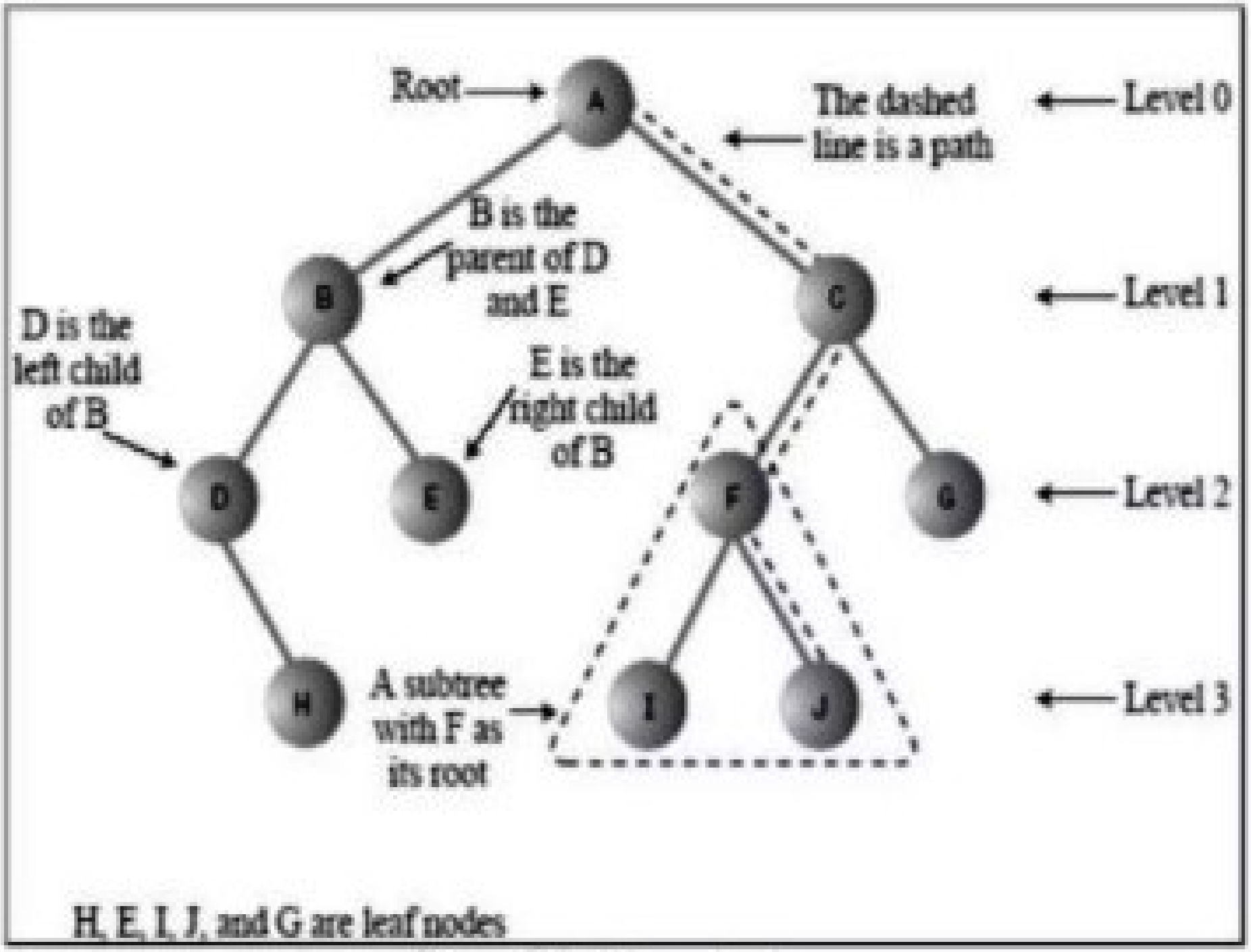

Suresh
Sunita
Sameer
Sahil
Rahul
Tina
Mala
Pooja
Nikita
Raj
Rajeev

Tree

- A tree is an abstract model of a hierarchical structure that consists of nodes with a parent-child relationship.
  - Tree is a sequence of nodes
  - There is a starting node known as a root node
  - Every node other than the root has a parent node.
  - Nodes may have any number of children

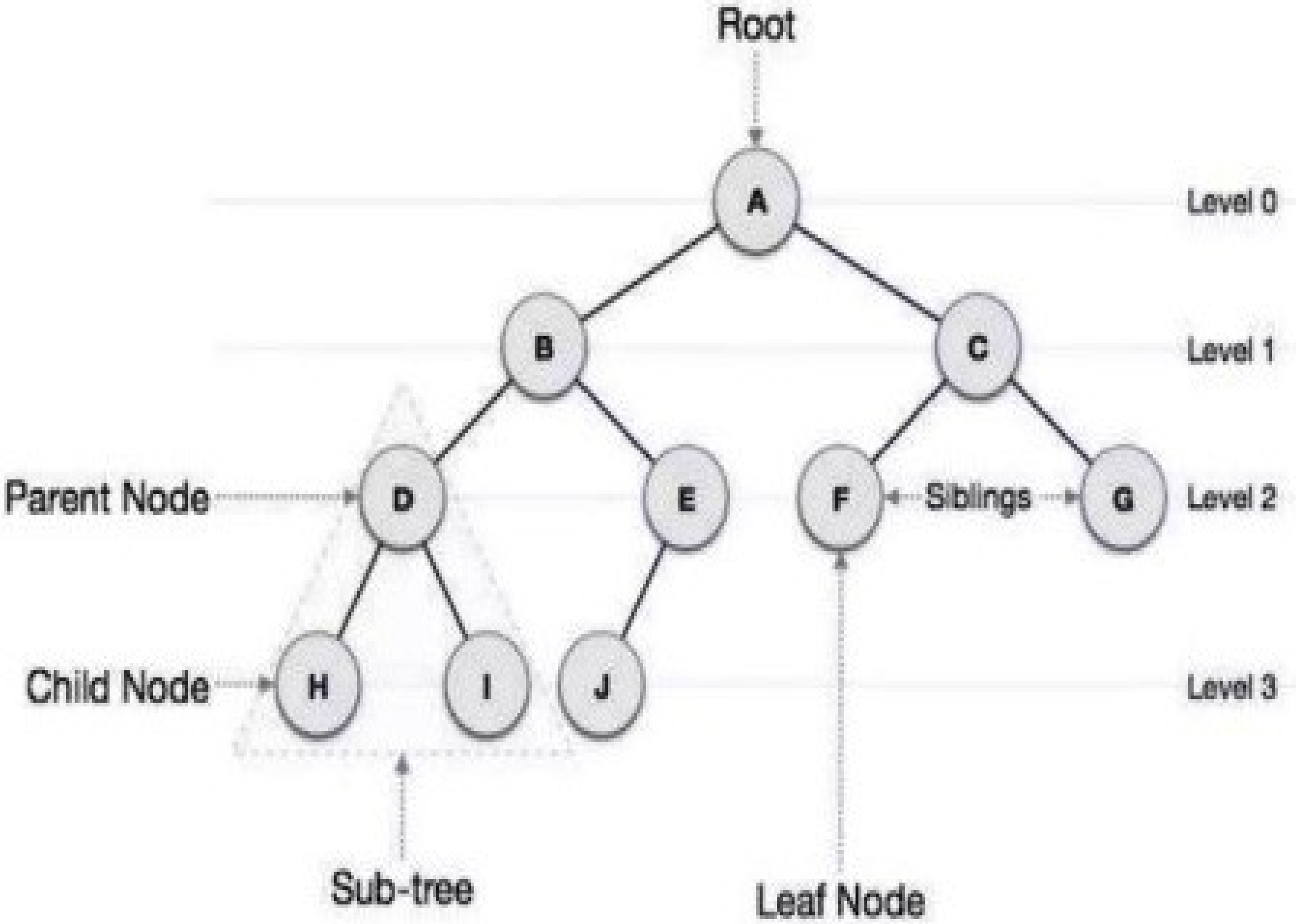


A has 3 children, B, C, D
A is parent of B


Root
A
The dashed line is a path
Level 0
B is the parent of D and E
B
C
Level 1
D is the left child of B
E is the right child of B
D
E
F
G
Level 2
H
A subtree with F as its root
I
J
Level 3
H, E, I, J, and G are leaf nodes

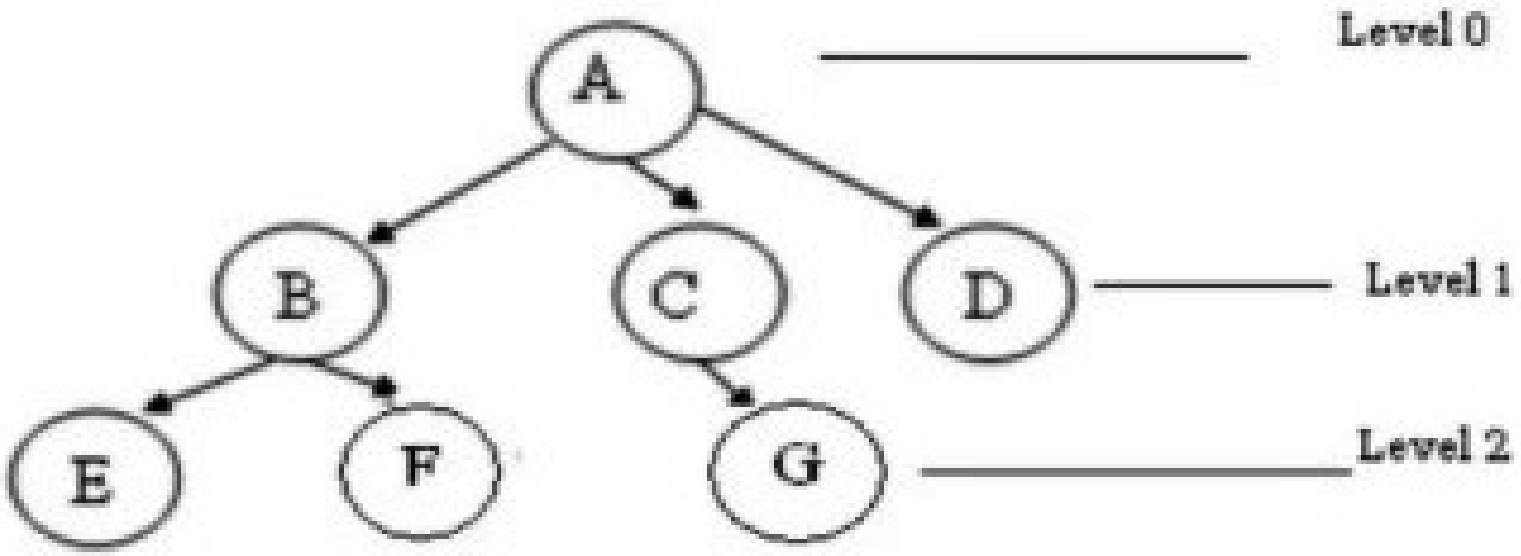
Some Key Terms:

- **Root** – Node at the top of the tree is called root.
- **Parent** – Any node except root node has one edge upward to a node called parent.
- **Child** – Node below a given node connected by its edge downward is called its child node.
- **Sibling** – Child of same node are called siblings
- **Leaf** – Node which does not have any child node is called leaf node.
- **Sub tree** – Sub tree represents descendants of a node.
- **Levels** – Level of a node represents the generation of a node. If root node is at level 0, then its next child node is at level 1, its grandchild is at level 2 and so on.
- **keys** – Key represents a value of a node based on which a search operation is to be carried out for a node.


Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Siblings
Child Node
Sub-tree
Leaf Node

Some Key Terms:

- Degree of a node:
  - The degree of a node is the number of children of that node
- Degree of a Tree:
  - The degree of a tree is the maximum degree of nodes in a given tree
- Path:
  - It is the sequence of consecutive edges from source node to destination node.
- Height of a node:
  - The height of a node is the max path length form that node to a leaf node.
- Height of a tree:
  - The height of a tree is the height of the root
- Depth of a tree:
  - Depth of a tree is the max level of any leaf in the tree



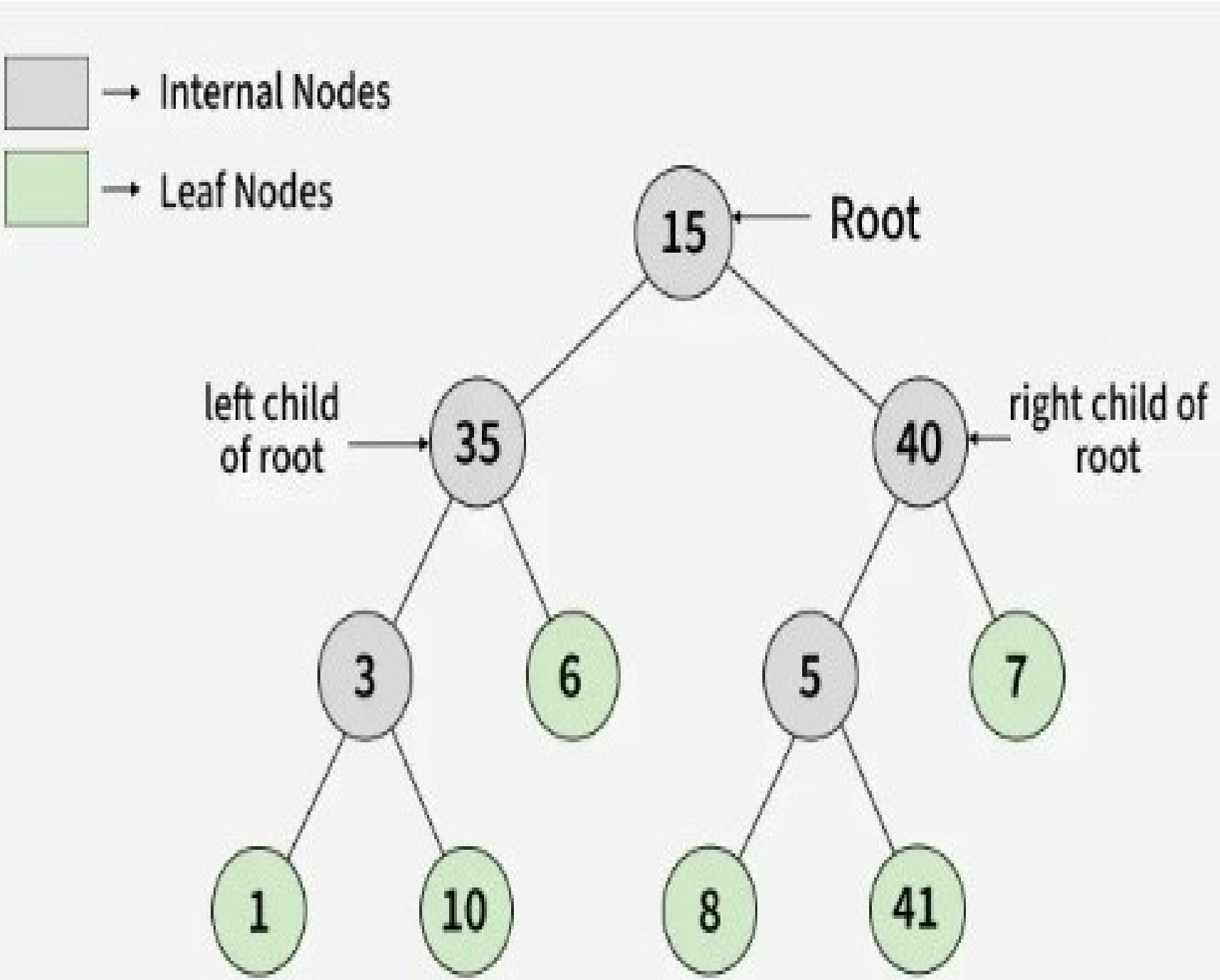
- ✔ A is the root node
- ✔ B is the parent of E and F
- ✔ D is the sibling of B and C
- ✔ E and F are children of B
- ✔ E, F, G, D are external nodes or leaves
- ✔ A, B, C are internal nodes
- ✔ Depth of F is 2
- ✔ the height of tree is 2
- ✔ the degree of node A is 3
- ✔ The degree of tree is 3

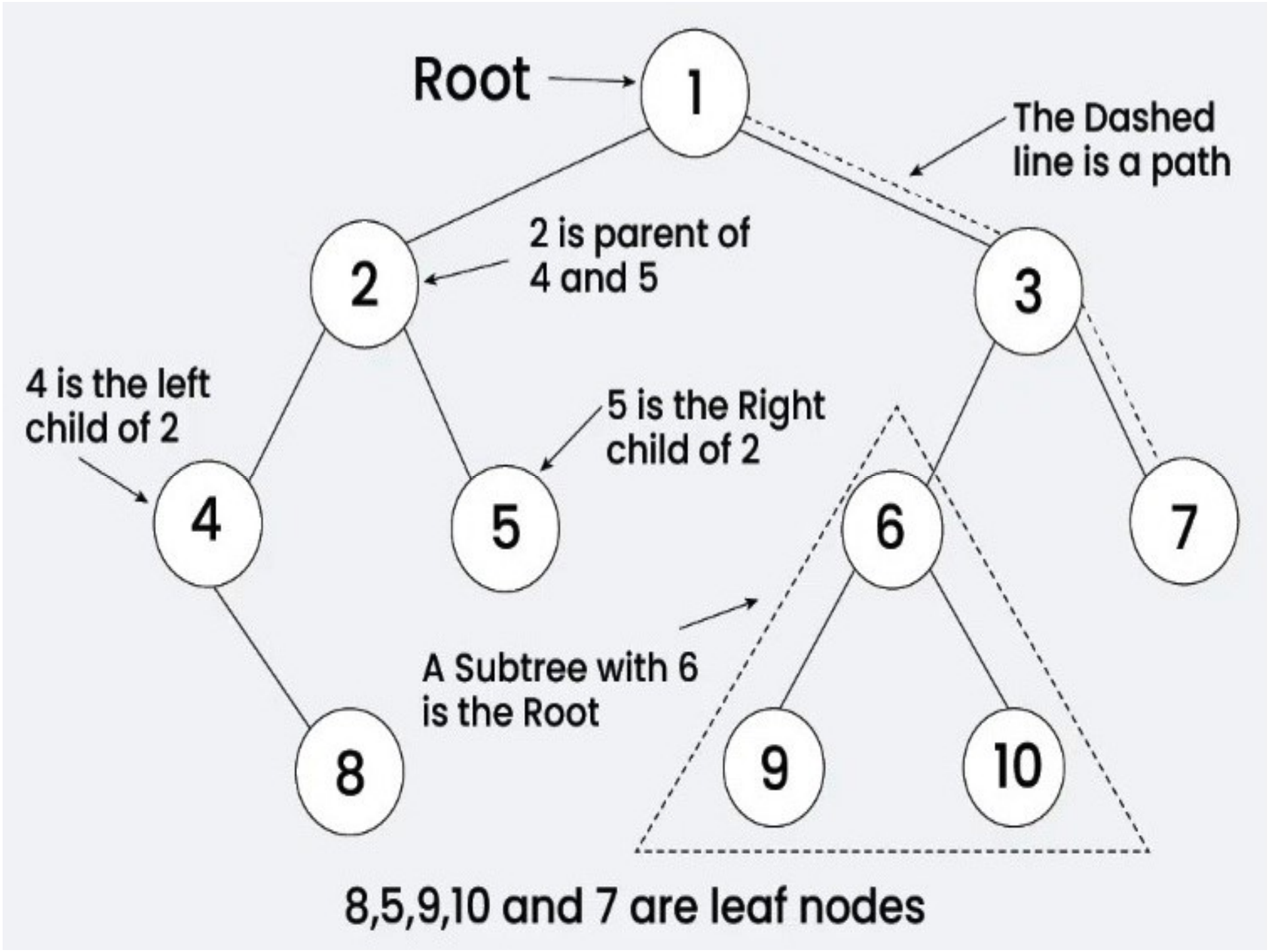
Characteristics of trees

- Non-linear data structure
- Combines advantages of an ordered array
- Searching as fast as in ordered array
- Insertion and deletion as fast as in linked list
- Simple and fast

Binary Tree

- Binary tree is a non-linear and hierarchical data structure where each node has **at most two children** referred to as the left child and the right child.
- The topmost node in a binary tree is called the root, and the bottom-most nodes(having no children) are called leaves.


Internal Nodes
Leaf Nodes
15
Root
left child of root
35
40
right child of root
3
6
5
7
1
10
8
41


Root
1
The Dashed
line is a path
2 is parent of
4 and 5
2
3
4 is the left
child of 2
5 is the Right
child of 2
4
5
6
7
A Subtree with 6
is the Root
8
9
10
8,5,9,10 and 7 are leaf nodes

What is binary search tree?

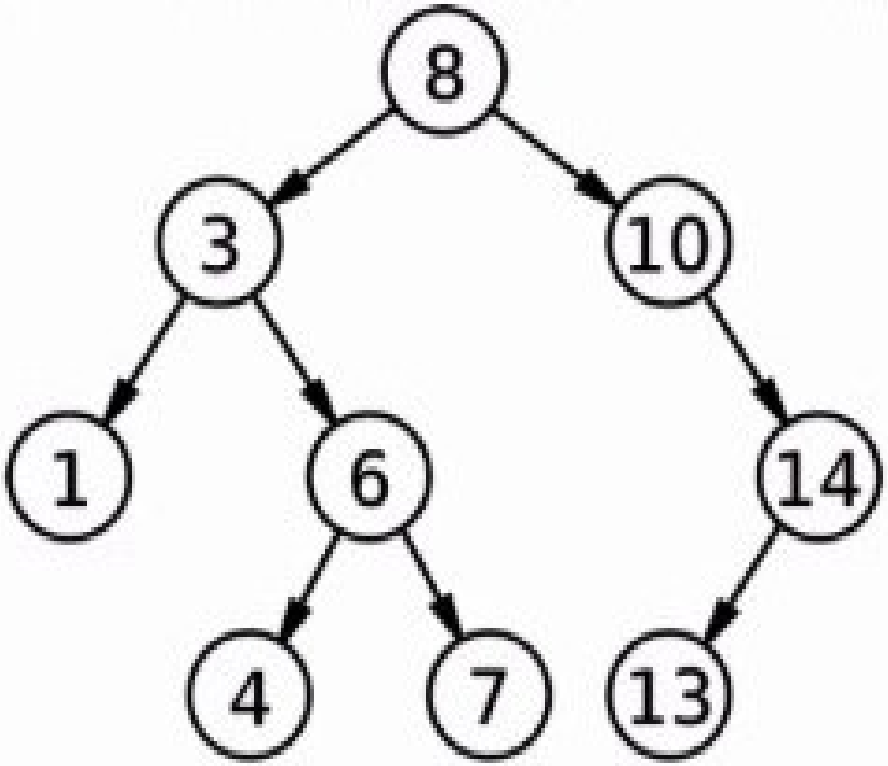
- Binary Search Tree is a binary tree in which every node contains only smaller values in its left subtree and only larger values in its right subtree.
- A Binary Search Tree (BST) is a rooted binary tree, whose nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted left and right. The tree should satisfy the BST property, which states that the key in each node must be greater than all keys stored in the left sub-tree, and not greater than all keys in the right sub-tree. Ideally, only unique values should be present in the tree.

Binary Tree vs. Binary Search Tree (BST)

Feature	Binary Tree	Binary Search Tree (BST)
Ordering	No specific order among nodes.	Nodes are arranged in sorted order: left children are smaller than the parent, and right children are larger.
Structure	Hierarchical, each node has at most two children.	A specialized binary tree with an organized, value-based structure.
Duplicate Values	Can easily accommodate duplicate values.	Typically does not accommodate duplicate values, or has specific rules for handling them.
Search Efficiency	Slower; typically requires searching the entire tree (O(n) time complexity).	Faster; the ordered structure allows for efficient searches, typically O(log n) time for balanced trees.
Primary Use Case	Representing natural hierarchies (e.g., file systems, organizational charts, expression trees in compilers).	Applications requiring frequent and fast data search, insertion, and deletion (e.g., database indexing, symbol tables, dictionaries).

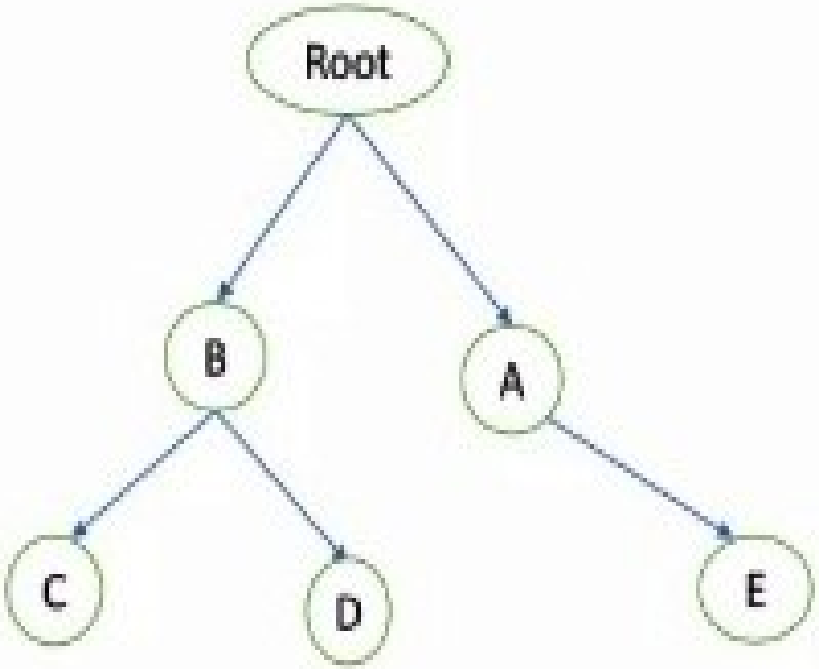
Properties of Binary Search Tree

- In a binary search tree, all the nodes in the left subtree of any node contains smaller values and all the nodes in the right subtree of any node contains larger values as shown in the following figure...



Properties of Binary Search Tree

- a unique path exists from the root to every other node



Operations in Binary Search Tree

- The following basic operations are performed on a binary search tree.
  1. Creation
  2. Traversing
     - i. Pre-order
     - ii. Post-Order
     - iii. In-order
  3. Search
  4. insertion
  5. Deletion
  6. To find minimum number.
  7. To find maximum number.

Creation of BST

• Creating a **Binary Search Tree (BST)** involves organizing data into a hierarchical structure where each node follows a specific ordering rule: the left child's value is smaller than the parent, and the right child's value is larger.

BST Property

For any node in the tree:

Left Subtree: Contains only nodes with values **less than** the node's value.

Right Subtree: Contains only nodes with values **greater than** the node's value.

Recursive Rule: Both the left and right subtrees must themselves be valid binary search trees.

Steps to Create a BST

- Building a BST typically involves starting from an empty root and inserting elements one by one.
- **Initialize the Root**: If the tree is empty, the first value inserted becomes the **root node**.
- **Compare Subsequent Values**: For every new value, start at the root and compare it to the current node's value.
  - If the new value is **smaller**, move to the **left child**.
  - If the new value is **larger**, move to the **right child**

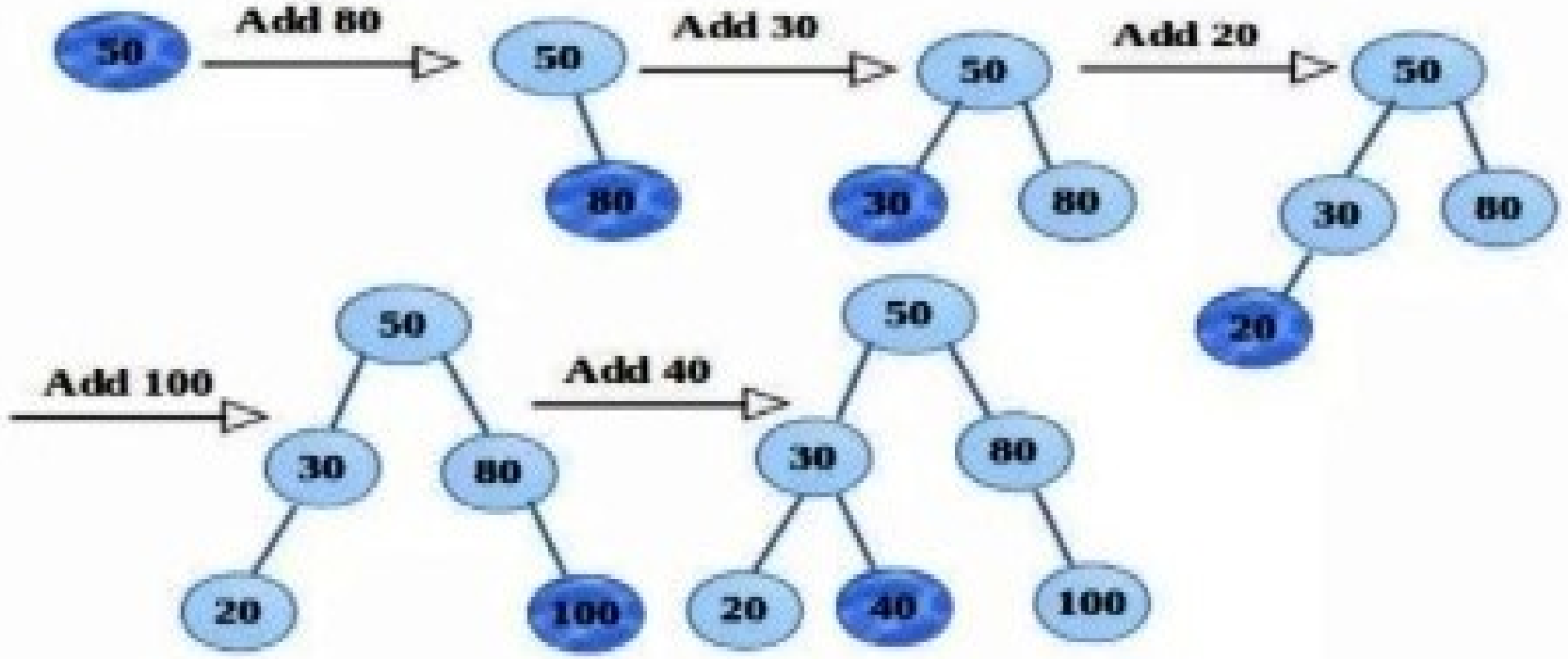
Steps to Create a BST Cont..

- **Repeat Until Empty Spot**: Continue this comparison recursively (moving down level by level) until you reach a null or empty position.
- **Insert the Node**: Place the new value at that unoccupied spot as a new **leaf node**.

Creation

- Let's take an example to create binary search tree by inserting the following elements

  50,80,30,20,100 and 40.



Common Traversal Methods

- In a **Binary Search Tree (BST)**, traversal refers to the process of visiting every node in the tree exactly once. Because of the BST's specific property—where left children are smaller and right children are larger than their parent—different traversal orders yield unique and useful results.

Common Traversal Methods

- Traversals are broadly categorized into **Depth-First Search (DFS)** and **Breadth-First Search (BFS)**.
- **1. In-order Traversal (Left, Root, Right)**
- This is the most significant traversal for BSTs. It visits the left subtree, then the root, and finally the right subtree. Codecademy +3
- **Result:** It returns the node values in **ascending sorted order**.
- **Use Cases:** Validating if a tree is a BST, finding the
- -th smallest element, or generating a sorted list of data. Codecademy +2
- **2. Pre-order Traversal (Root, Left, Right)**
- This method visits the root node first, followed by the left subtree and then the right subtree. W3Schools +1
- **Use Cases:** Creating a copy of the tree or generating a **prefix expression** (Polish notation) from an expression tree. OpenDSA +1
- **3. Post-order Traversal (Left, Right, Root)**
- It traverses the left and right subtrees completely before visiting the root node. W3Schools +1
- **Use Cases: Deleting a tree** (safely deleting children before parents) and generating a **postfix expression** (Reverse Polish notation).