# Trees

Akash Hegde

Seventh Sense Talent Solutions

Vivekananda Institute of Technology

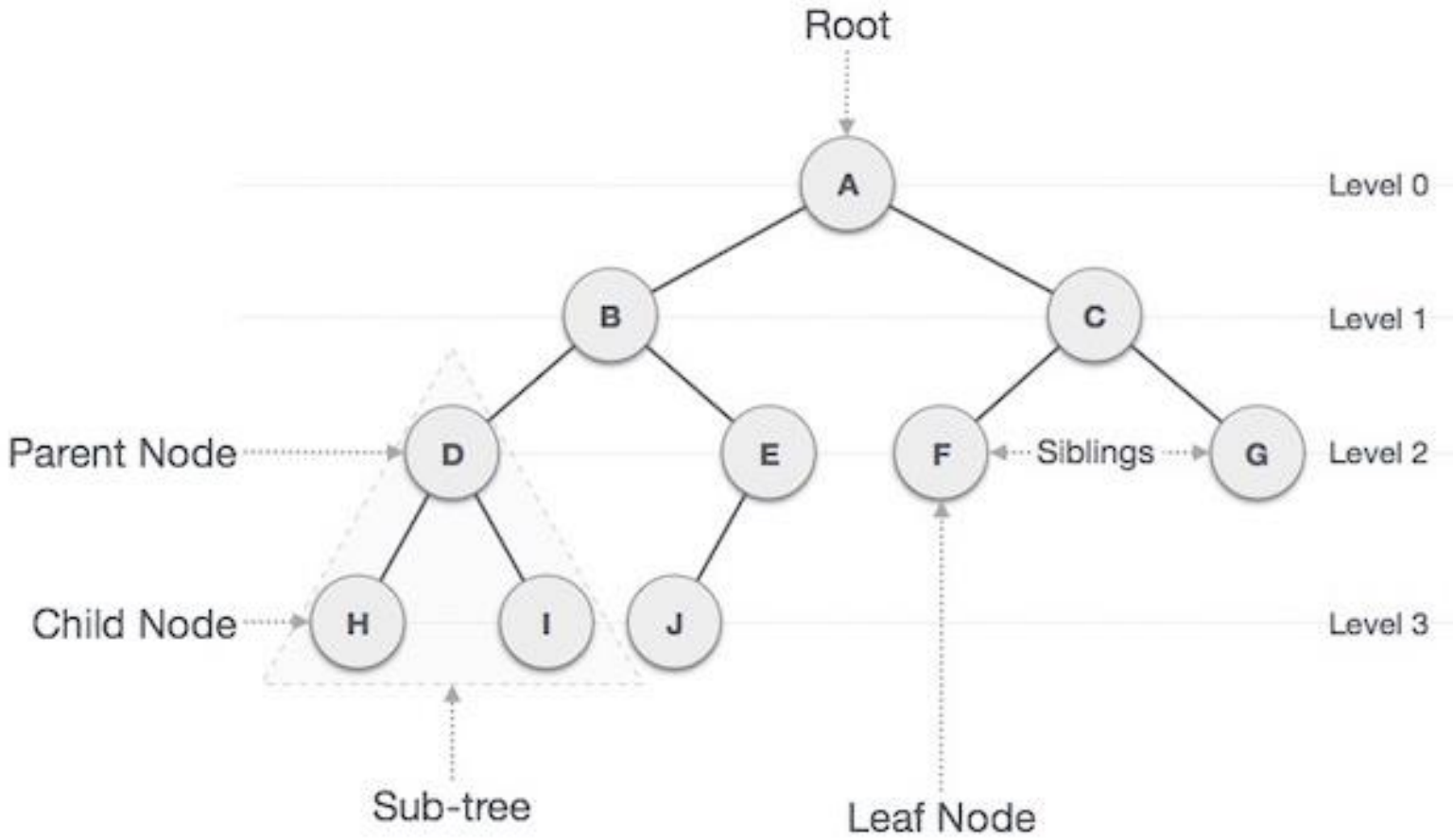
26 March 2021

# Introduction to Tree

- Tree represents the nodes connected by edges.
- We will discuss binary tree or binary search tree specifically.
- Binary Tree is a special data structure used for data storage purposes.
- A binary tree has a special condition that each node can have a maximum of two children.
- A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operation are as fast as in linked list.

# Introduction to Tree

- Representation of a tree –



# Tree Terminology

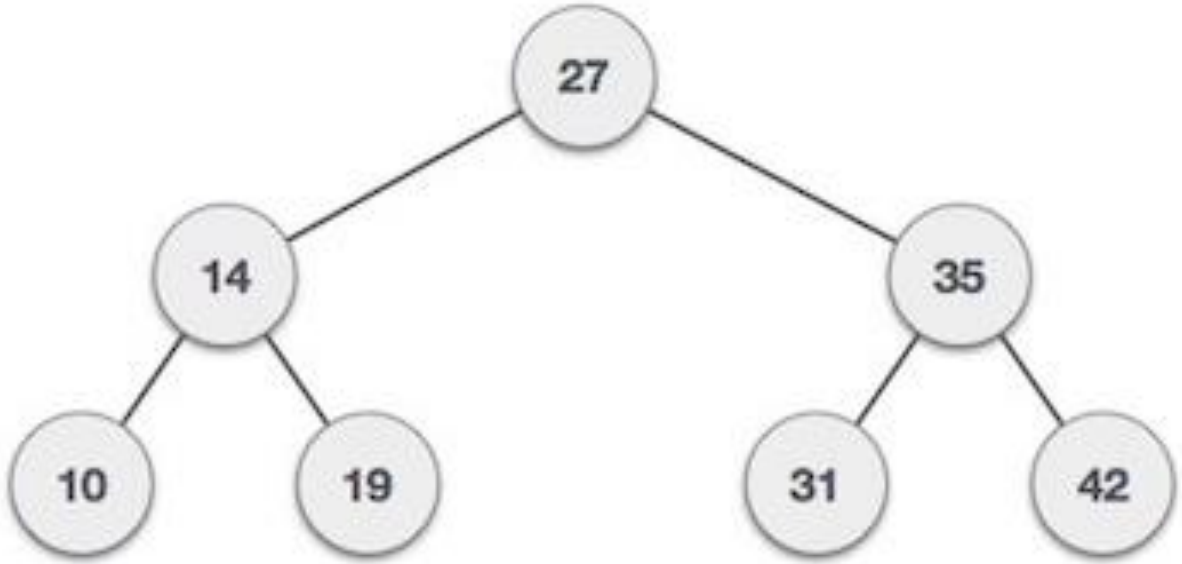
- **Path** – Path refers to the sequence of nodes along the edges of a tree.
- **Root** – The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- **Parent** – Any node except the root node has one edge upward to a node called parent.
- **Child** – The node below a given node connected by its edge downward is called its child node.
- **Leaf** – The node which does not have any child node is called the leaf node.

# Tree Terminology

- **Subtree** – Subtree represents the descendants of a node.
- **Visiting** – Visiting refers to checking the value of a node when control is on the node.
- **Traversing** – Traversing means passing through nodes in a specific order.
- **Levels** – Level of a node represents the generation of a node. If the 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.

# Binary Search Tree – Introduction

- Binary Search Tree exhibits a special behavior.
- A node's left child must have a value less than its parent's value and the node's right child must have a value greater than its parent value.



# Binary Search Tree – Implementation

- Tree is represented as a node.
- In a tree, all nodes share common construct.
- struct node {
  int data;
  struct node *leftChild;
  struct node *rightChild;
  };

# Binary Search Tree – Operations

- **Insert** – Inserts an element in a tree, OR, create a tree.
- **Search** – Searches an element in a tree.
- **Preorder Traversal** – Traverses a tree in a pre-order manner.
- **Inorder Traversal** – Traverses a tree in an in-order manner.
- **Postorder Traversal** – Traverses a tree in a post-order manner.

# Binary Search Tree – Insert

- The very first insertion creates the tree.
- Afterwards, whenever an element is to be inserted, first locate its proper location.
- Start searching from the root node, then if the data is less than the key value, search for the empty location in the left subtree and insert the data.
- Otherwise, search for the empty location in the right subtree and insert the data.

# Binary Search Tree – Insert

- Algorithm:

```
If root is NULL
        then create root node
return
If root exists then
        compare the data with node.data
        while until insertion position is located
            If data is greater than node.data
                goto right subtree
            else
                goto left subtree
        endwhile
        insert data
end If
```

# Binary Search Tree – Search

- Whenever an element is to be searched, start searching from the root node, then if the data is less than the key value, search for the element in the left subtree.
- Otherwise, search for the element in the right subtree.
- Follow the same algorithm for each node.

# Binary Search Tree – Search

- Algorithm:

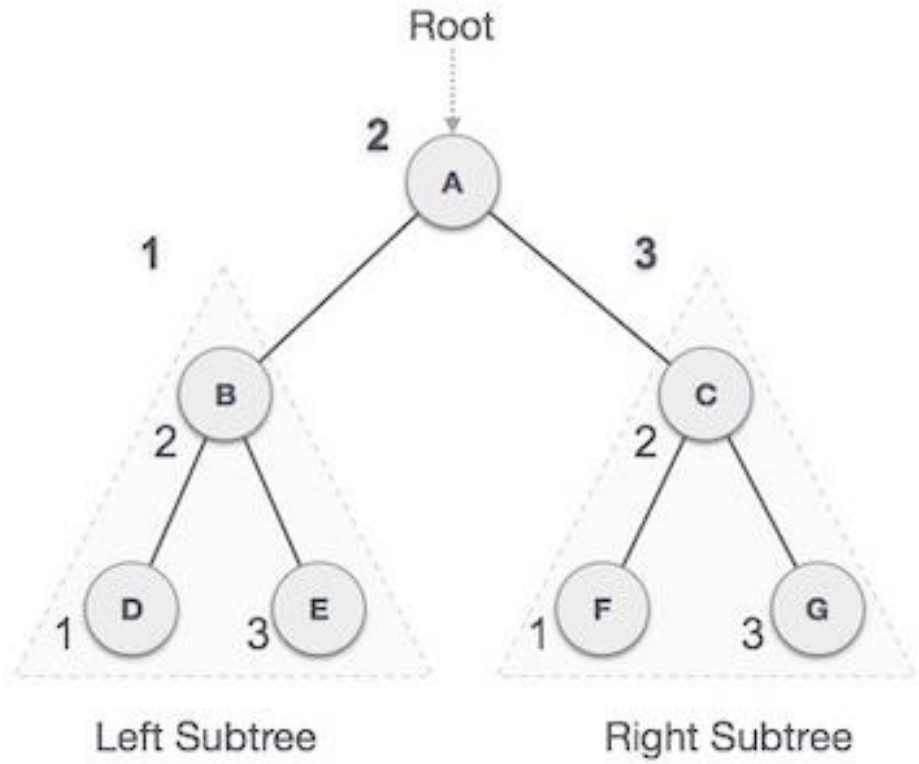
```
If root.data is equal to search.data
        return root
else
        while data not found
                If data is greater than node.data
                        goto right subtree
                else
                        goto left subtree
                If data found
                        return node
        endwhile
        return data not found
end if
```

# Binary Search Tree – Tree Traversal

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- Because, all nodes are connected via edges (links) we always start from the root (head) node.
- That is, we cannot randomly access a node in a tree.
- There are three ways which we use to traverse a tree –
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal
- Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

# Binary Search Tree – In-order Traversal

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree.
- Every node may represent a subtree itself.

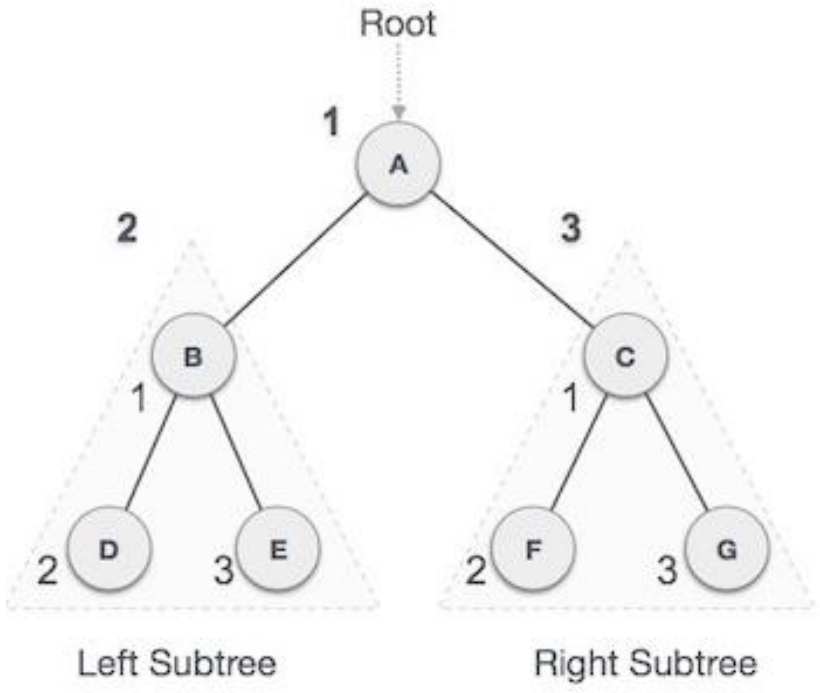


# Binary Search Tree – In-order Traversal

- We start from **A**, and following in-order traversal, we move to its left subtree **B**.
- **B** is also traversed in-order.
- The process goes on until all the nodes are visited.
- The output of inorder traversal of this tree will be – $D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$
- Algorithm:
  Until all nodes are traversed –
  **Step 1** – Recursively traverse left subtree.
  **Step 2** – Visit root node.
  **Step 3** – Recursively traverse right subtree.

# Binary Search Tree – Pre-order Traversal

- In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.
- Every node may represent a subtree itself.
- If a binary tree is traversed **pre-order**, the output will produce sorted key values in an ascending order.

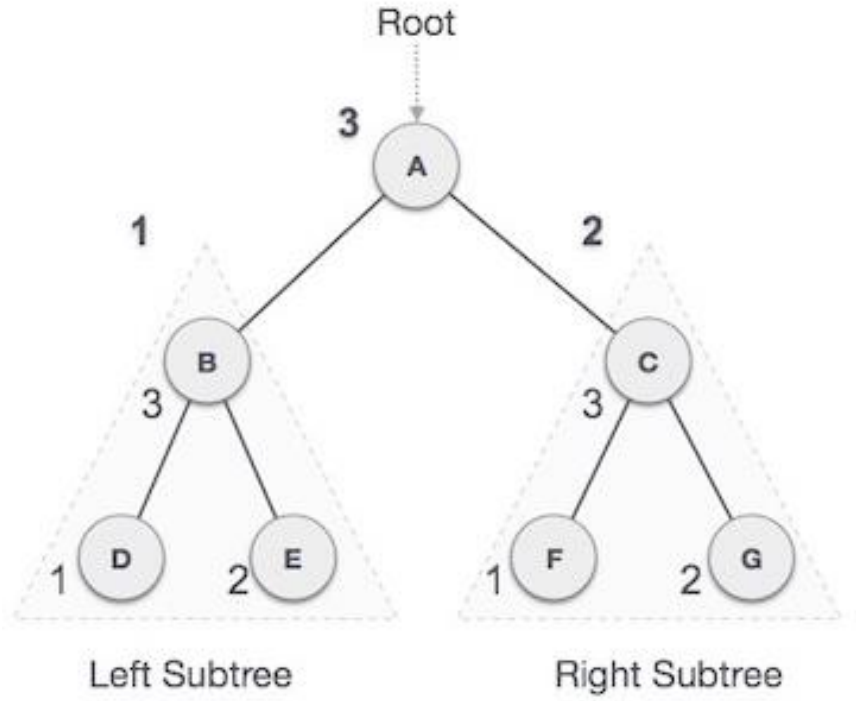


# Binary Search Tree – Pre-order Traversal

- We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**.
- **B** is also traversed pre-order.
- The process goes on until all the nodes are visited.
- The output of pre-order traversal of this tree will be – $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$
- Algorithm:
  Until all nodes are traversed –
  **Step 1** – Visit root node.
  **Step 2** – Recursively traverse left subtree.
  **Step 3** – Recursively traverse right subtree.

# Binary Search Tree – Post-order Traversal

- In this traversal method, the root node is visited last, hence the name.
- First we traverse the left subtree, then the right subtree and finally the root node.
- Every node may represent a subtree itself.

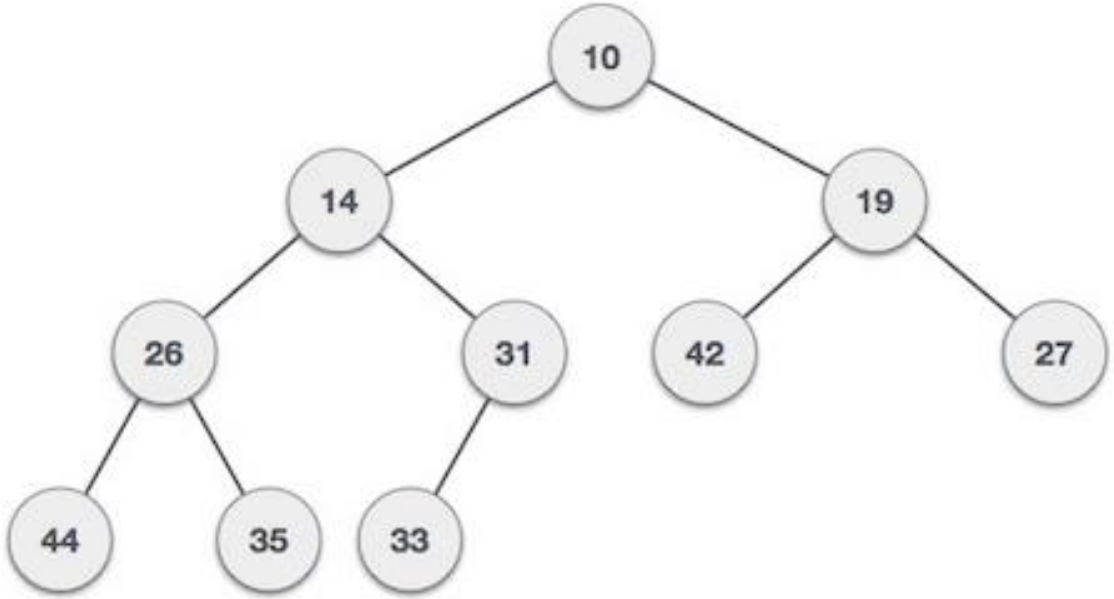


# Binary Search Tree – Post-order Traversal

- We start from **A**, and following Post-order traversal, we first visit the left subtree **B**.
- **B** is also traversed post-order.
- The process goes on until all the nodes are visited.
- The output of post-order traversal of this tree will be – $D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$
- Algorithm:
  Until all nodes are traversed –
  **Step 1** – Recursively traverse left subtree.
  **Step 2** – Recursively traverse right subtree.
  **Step 3** – Visit root node.

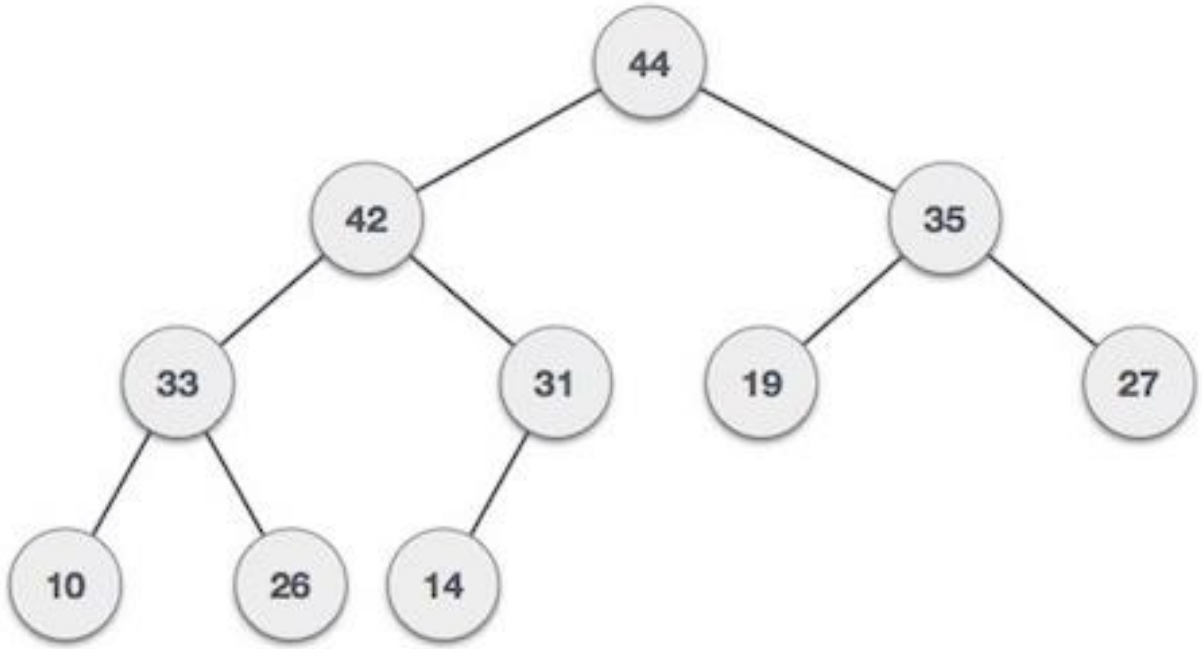
# Heap

- Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly.
- **Min-Heap** – Where the value of the root node is less than or equal to either of its children.



# Heap

- **Max-Heap** – Where the value of the root node is greater than or equal to either of its children.



Thank you!