Welcome!

Are you excited for a fun learning session?

Trees

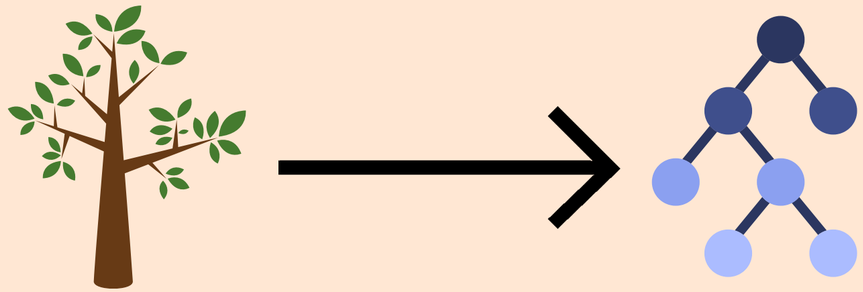
INTRODUCTION

A General Tree

- Each node may have zero or more children
- A binary tree is a specialized case of a general tree
- General trees are used to model applications such as file systems.

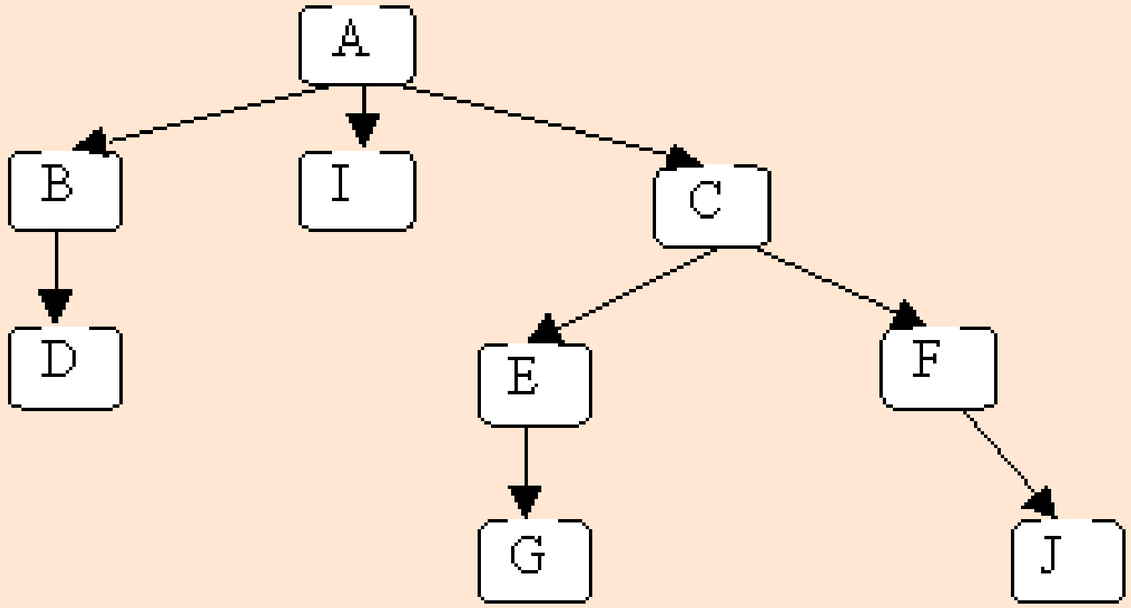
A Binary Tree

- Each node can have no more than two children
- Good way to understand trees



Terminology

- Each letter represents one **node**
- The arrows from one node to another are called **edges**
- The topmost node (with no incoming edges) is the **root** like (node A)
- The nodes with no children are called **leaves** like (nodes D, I, G & J);

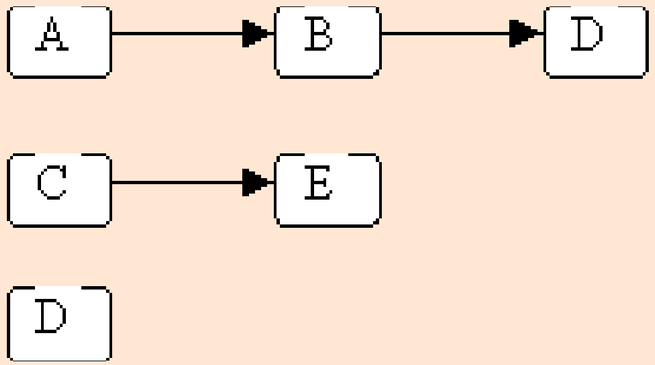


- Node A is called the parent
- Node B is called the child
- And the relation continues ...

Path

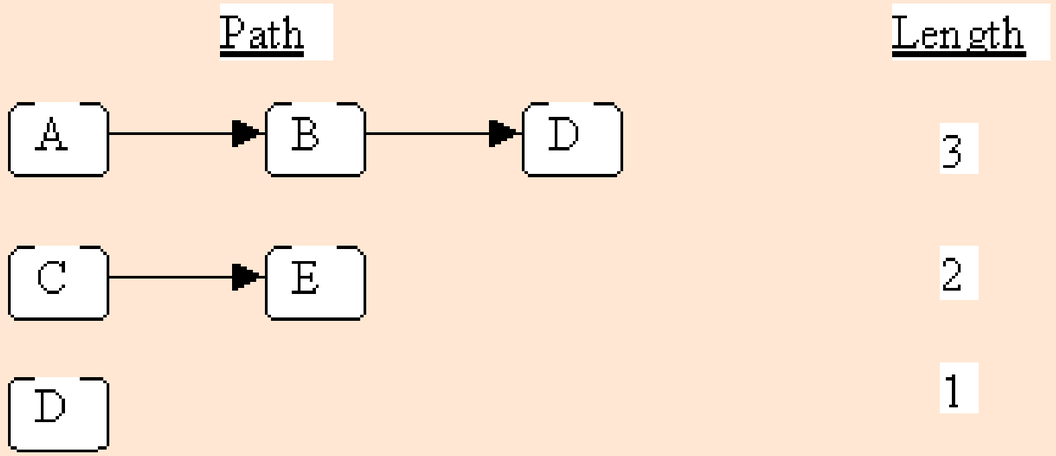
- a sequence of (zero or more) connected nodes

- Here are 3 of the paths in the tree shown above:



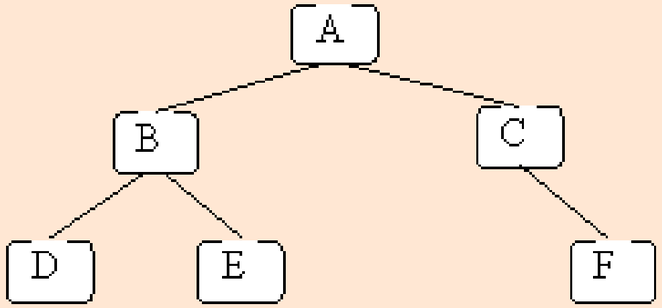
Length

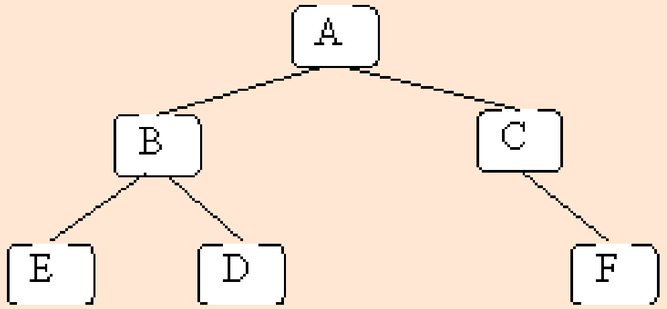
- A path in a tree is a sequence of (zero or more) connected nodes
- Here are 3 of the paths in the tree shown above:



Binary Tree

- Each node has 0, 1, or 2 children.
- Each child is either a left child or a right child.



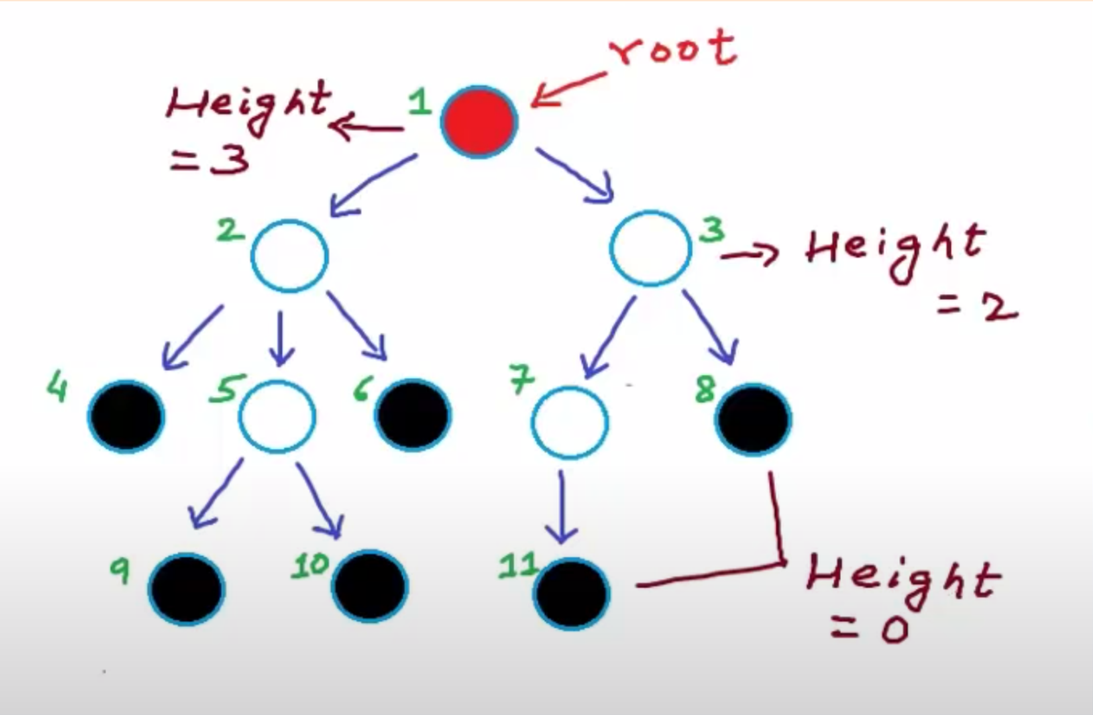


The two trees are different because the children of node B are different

Height

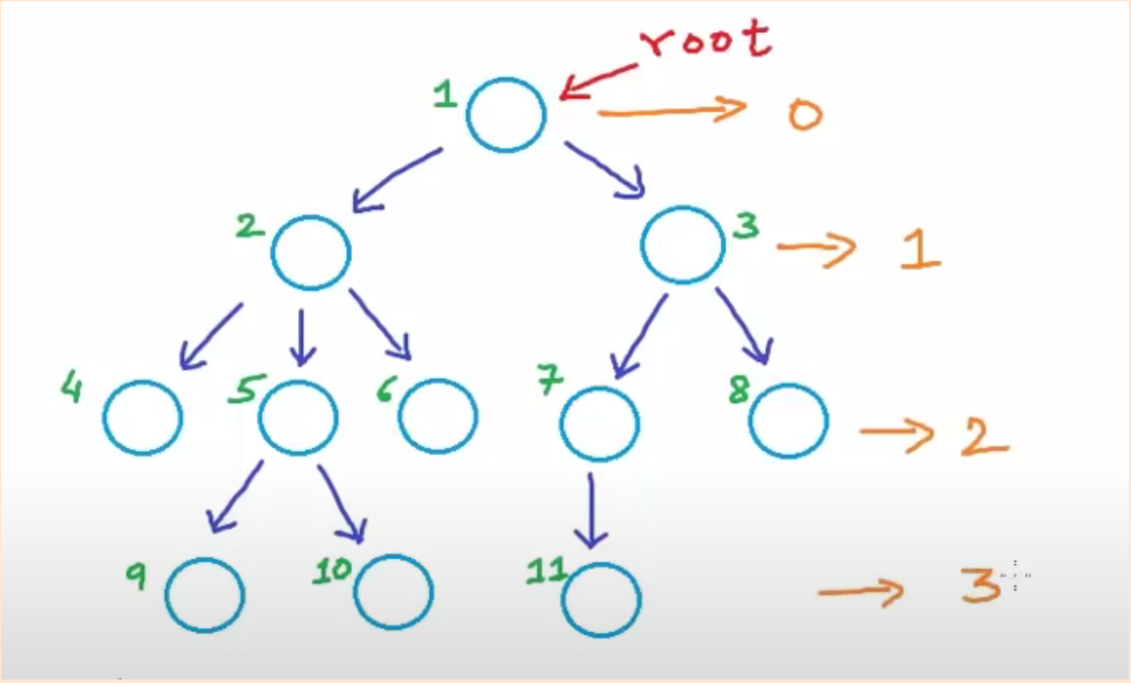
The height of a tree is the length of the longest path from the root to a leaf

- An empty tree has height = 0.



Depth

The depth of a node is the length of the path from the root to that node.



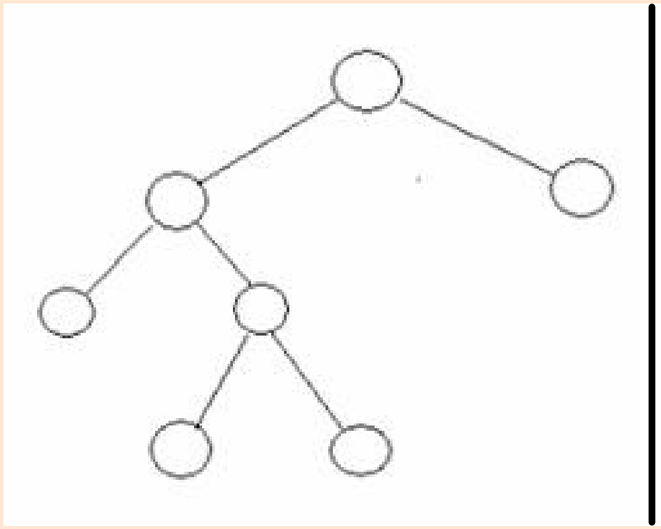
Full Binary Tree

- A binary tree in which each node has exactly zero or two children is called a full binary tree.
- In a full tree, there are no nodes with exactly one child.

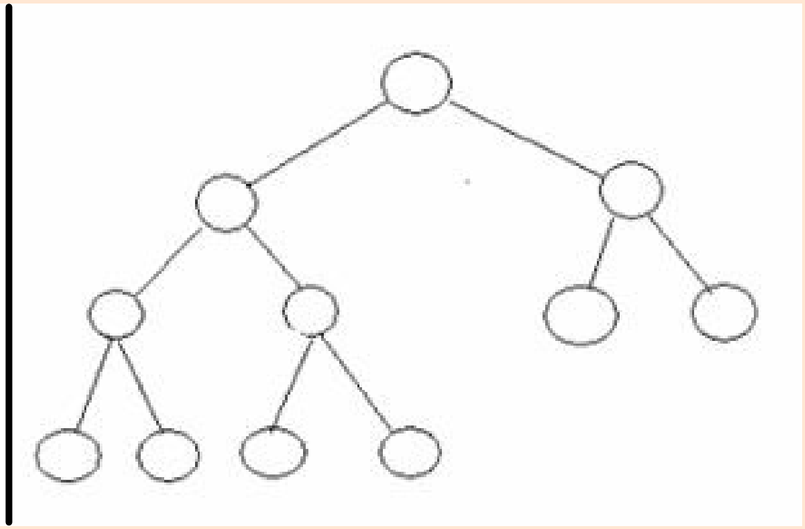
Complete Binary Tree

- Completely filled
- Possible exception of the bottom level, which is filled from left to right
- Height h has between 2h and 2(h+1)-1 nodes

Full Binary Tree



Complete Binary Tree

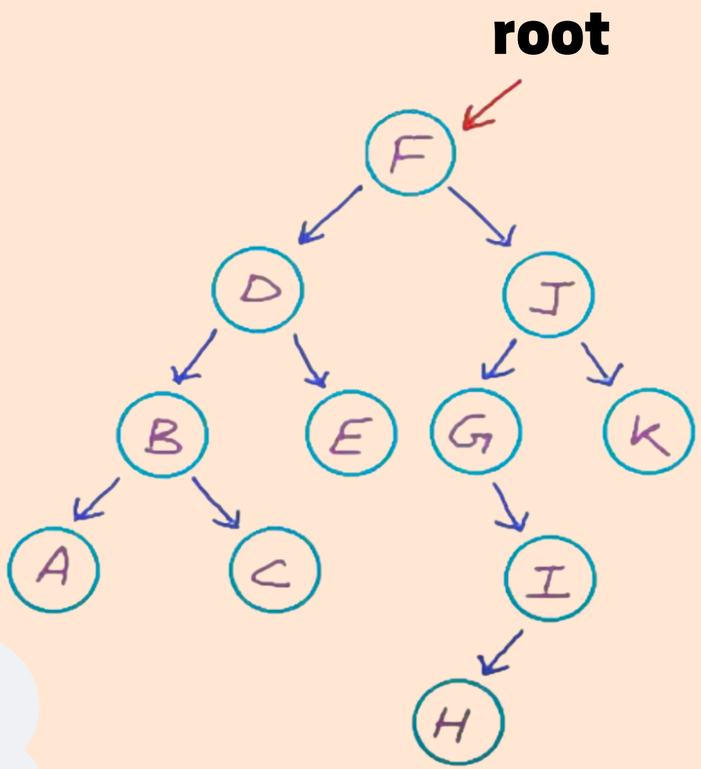


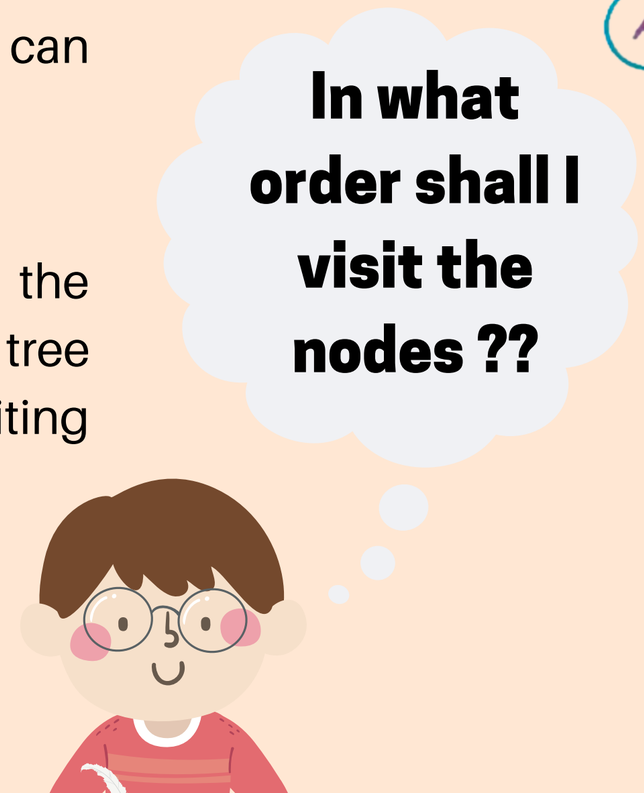
Tree Traversal

Tree is not a linear data structure, therefore there is no logical start or logical end.

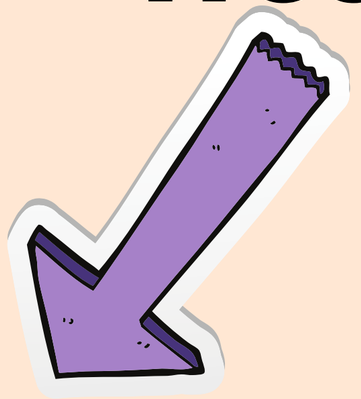
If we point to particular node then there are more than one directions in which one can move.

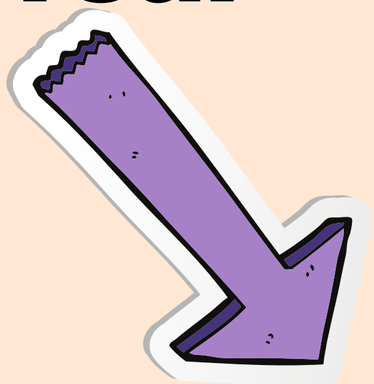
Formally, tree traversal is defined as the process of visiting each node in the tree exactly once in some order, for us visiting means printing.





Tree Traversal





Depth First Search

- Preorder
- Inorder
- Postorder

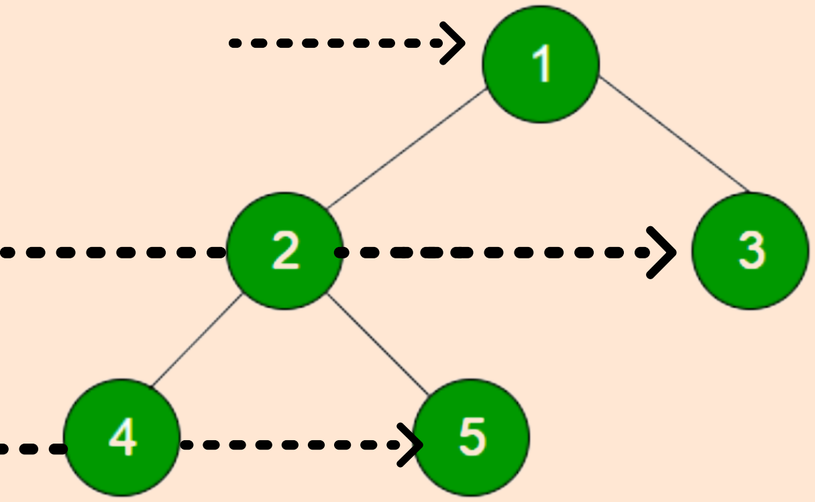
Breadth First Search

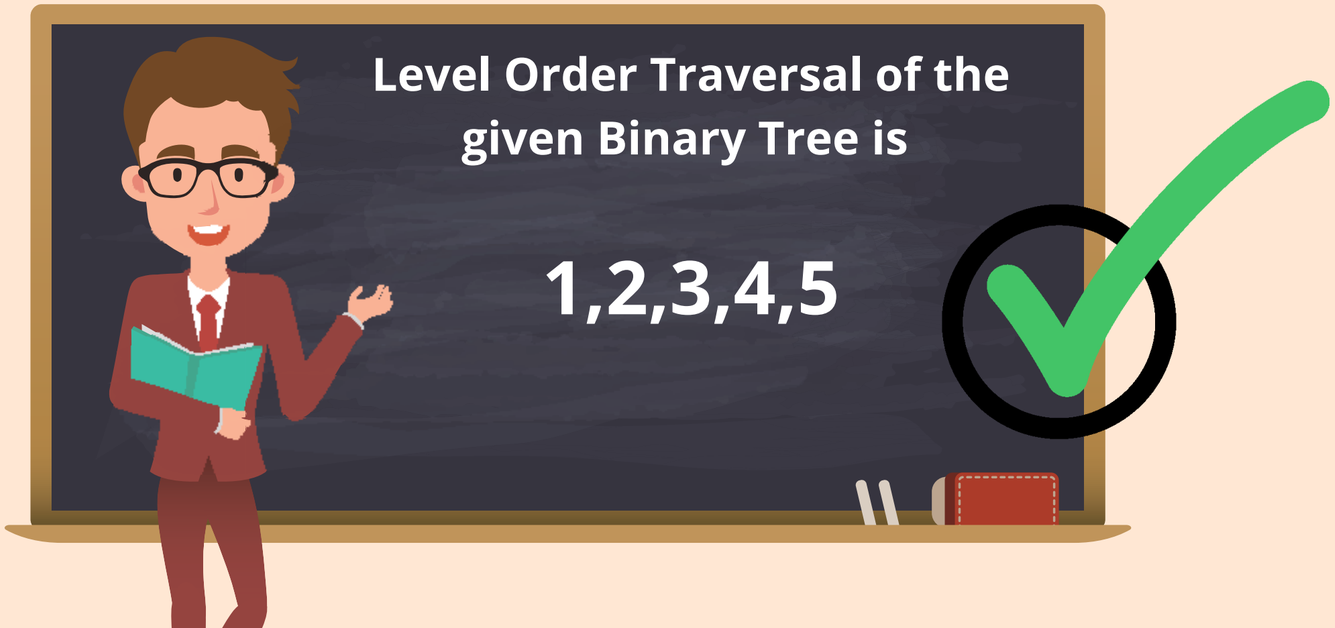
- Level Order Traversal

Level Order Traversal

Level order traversal of a tree is ***breadth first traversal*** for the tree.

We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.





Implementation

Implemented using queue data structure

Steps

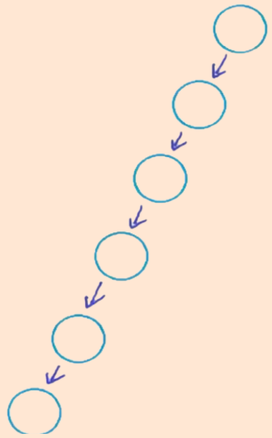
1. Push the root node into the queue
2. While the queue is not empty do steps 3 and 4
3. Remove the front node, print it
4. If left child exists, push it....If right child exists push it

Time Complexity

O(n) n - number of nodes in the tree

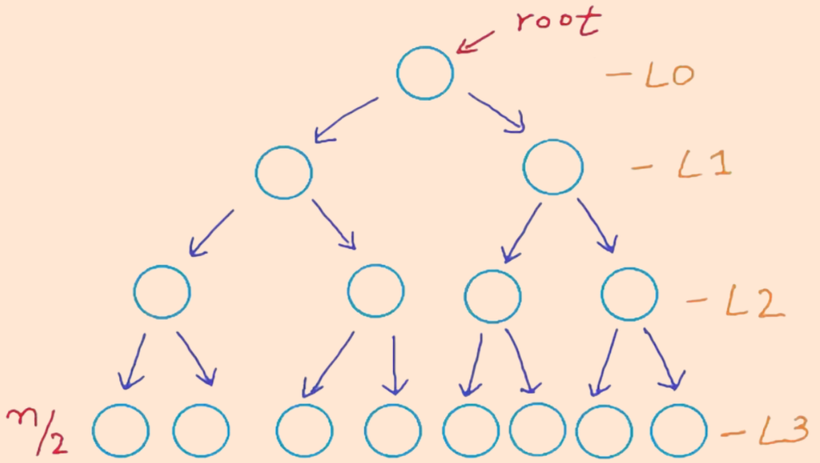
Space Complexity

Best - O(1)



Average - O(n) n - number of nodes in the tree

Worst - O(n) n - number of nodes in the tree



Depth First Search (DFS) Traversal

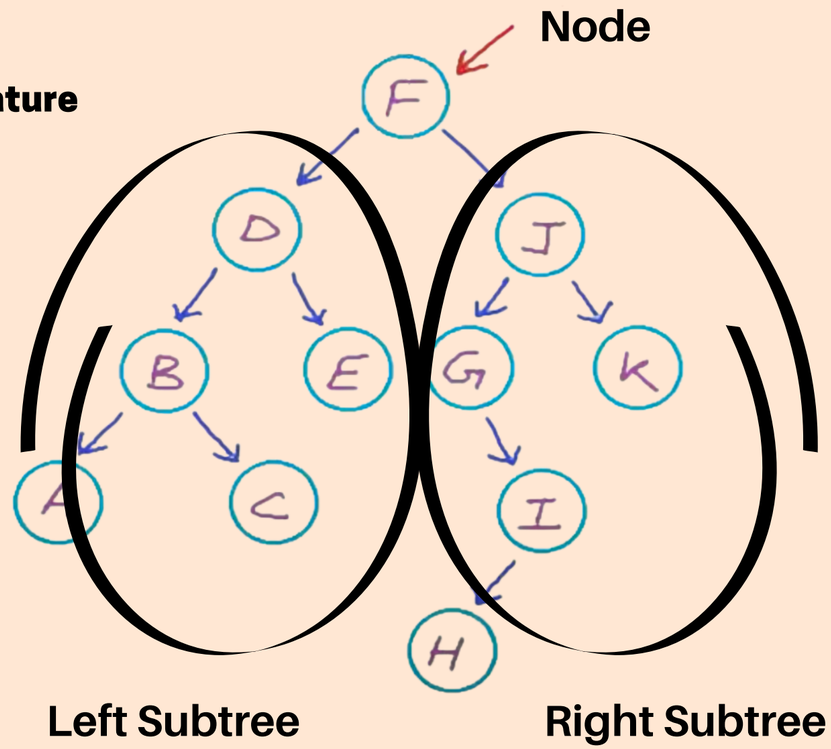
We scan the tree in a depth first manner

At each node, there are 2 directions go to => the order in which you need to traverse those directions determines the type DFS traversal you use.

1. Preorder
2. Inorder
3. Postorder

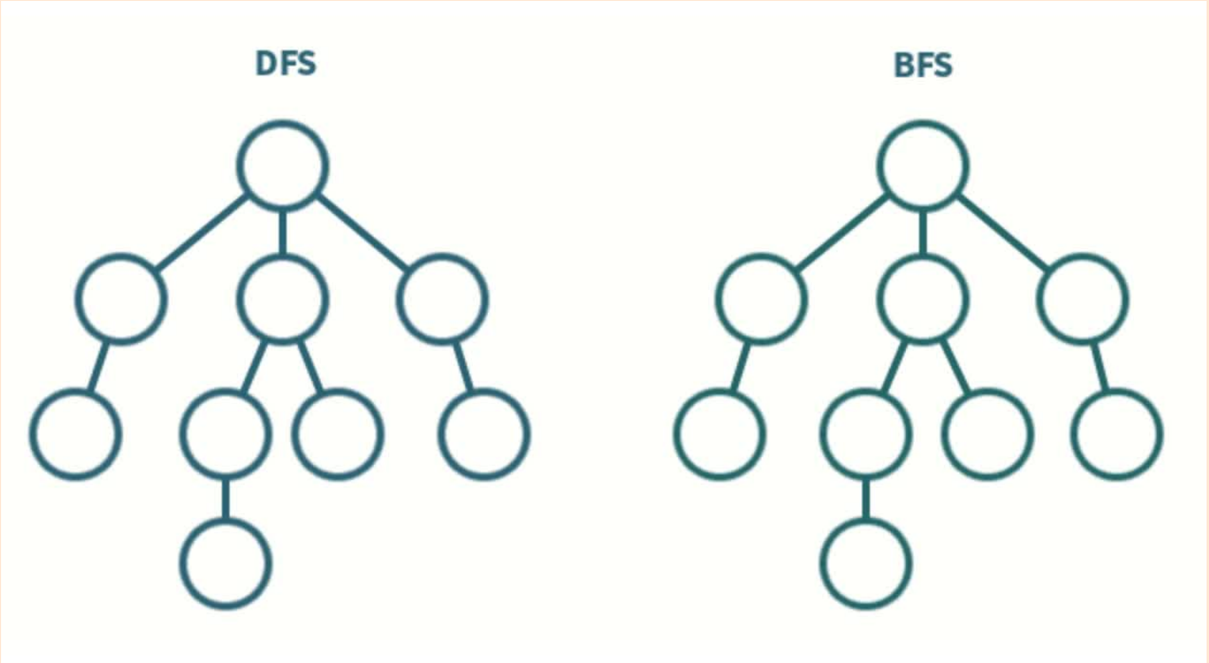
All differing in the relative order of visiting Left subtree, Right subtree and the Node itself.

Notice the similar nature of subproblems



DFS and BFS

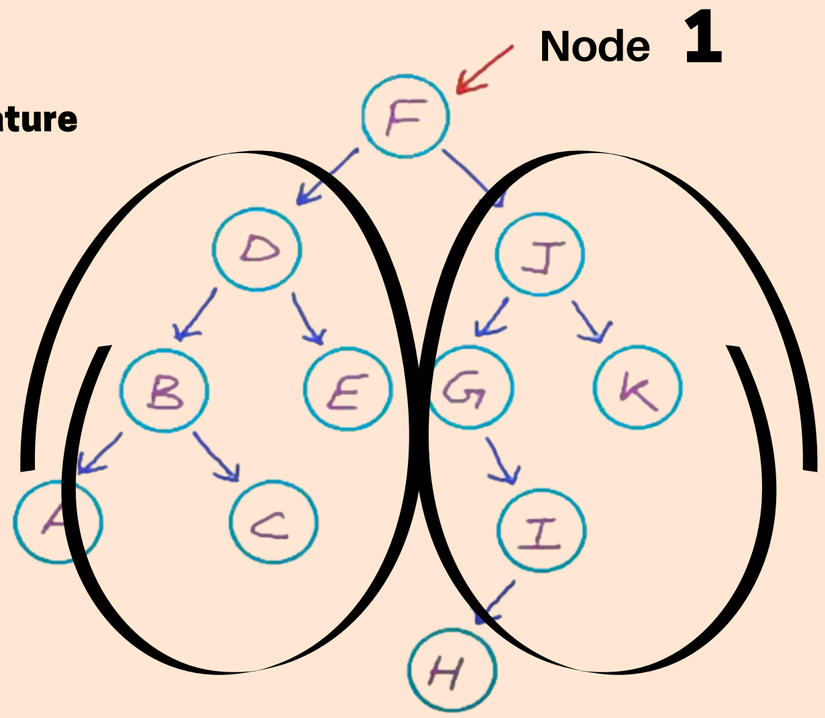
- DFS - scan tree in a depth first manner.
- BFS - scan tree in a level first manner.

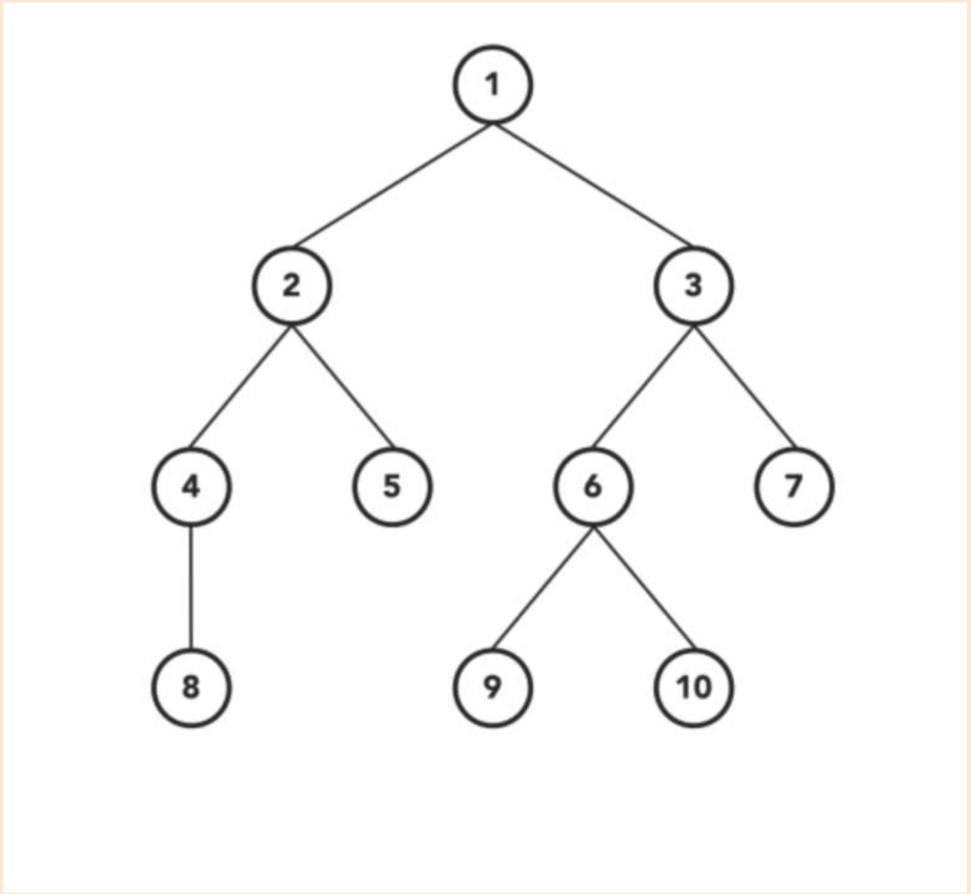


Preorder Traversal

Traverse each node of the tree in the order :
Visit Node, Left Subtree, Right Subtree

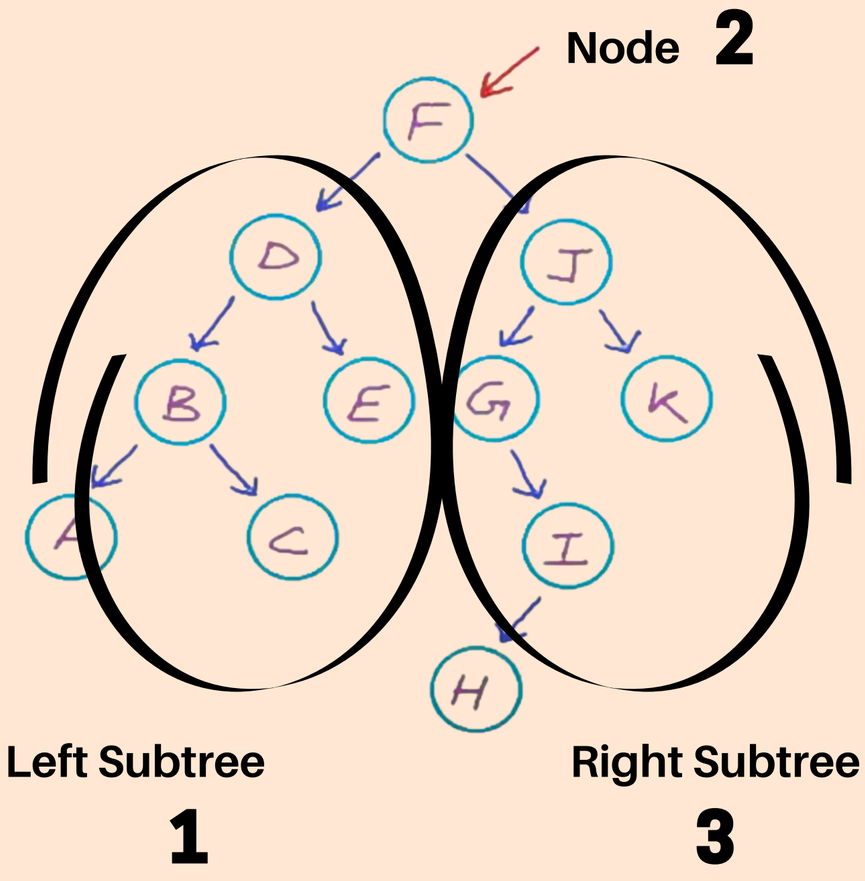
Notice the similar nature of subproblems

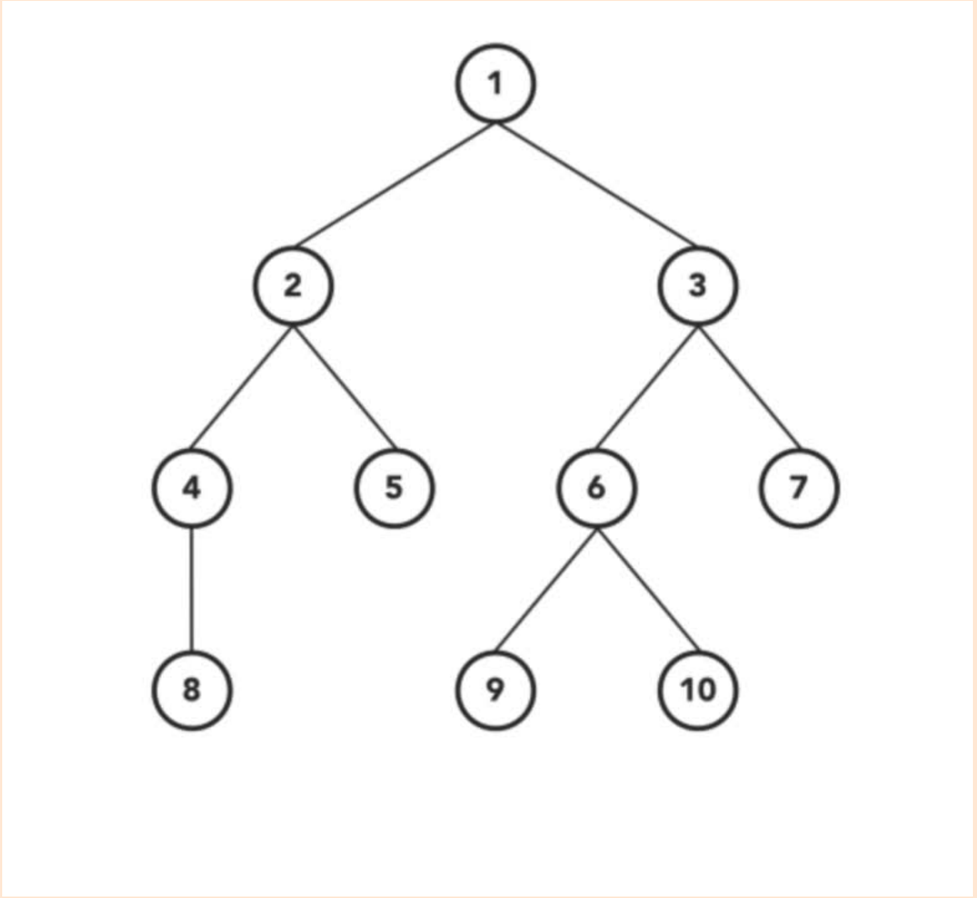




Inorder Traversal

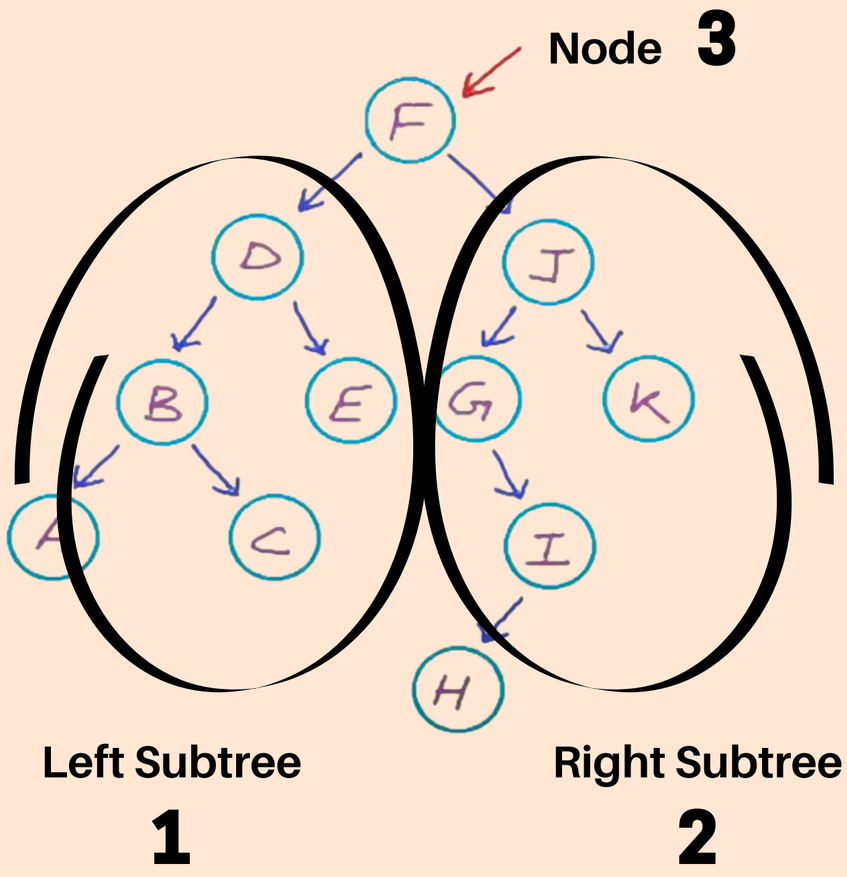
Traverse each node of the tree in the order :
Visit Left Subtree, Node, Right Subtree

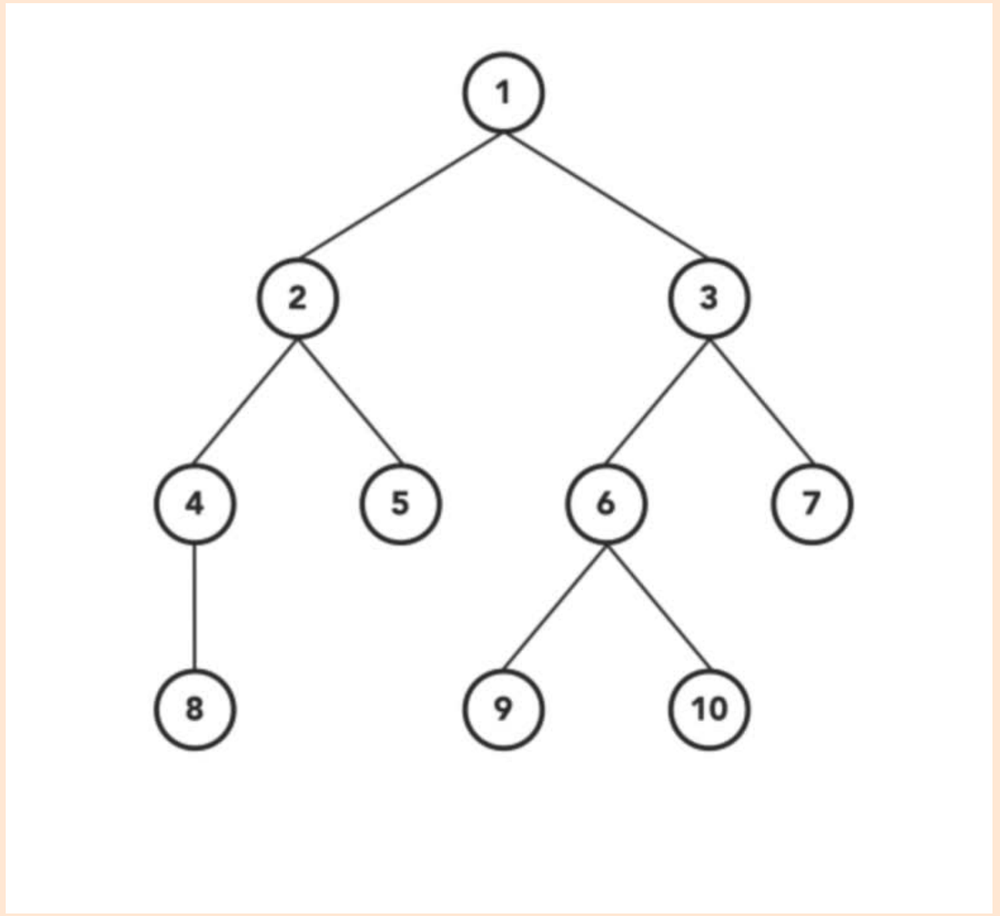




Postorder Traversal

Traverse each node of the tree in the order :
Visit Left Subtree, Right Subtree, Node





Let's Code

- **Preorder Traversal**
- **Inorder Traversal**
- **Postorder Traversal**

Vertical Order Traversal of a Binary Tree

https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/

Diagonal Order Traversal of a Binary Tree

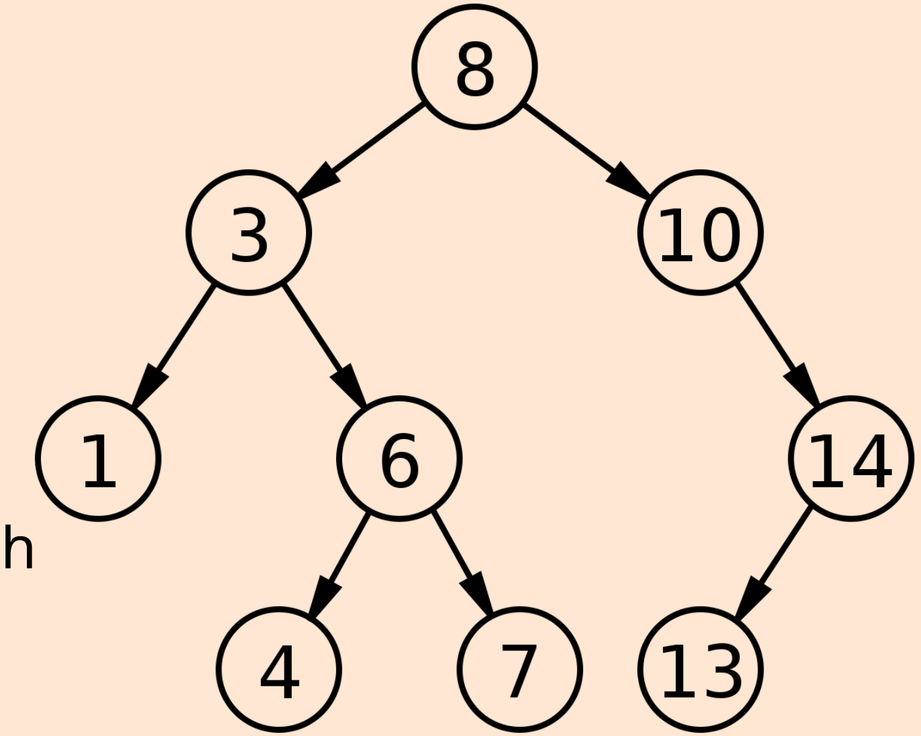
https://www.interviewbit.com/problems/diagonal-traversal/

Zigzag Order Traversal of a Binary Tree

https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/

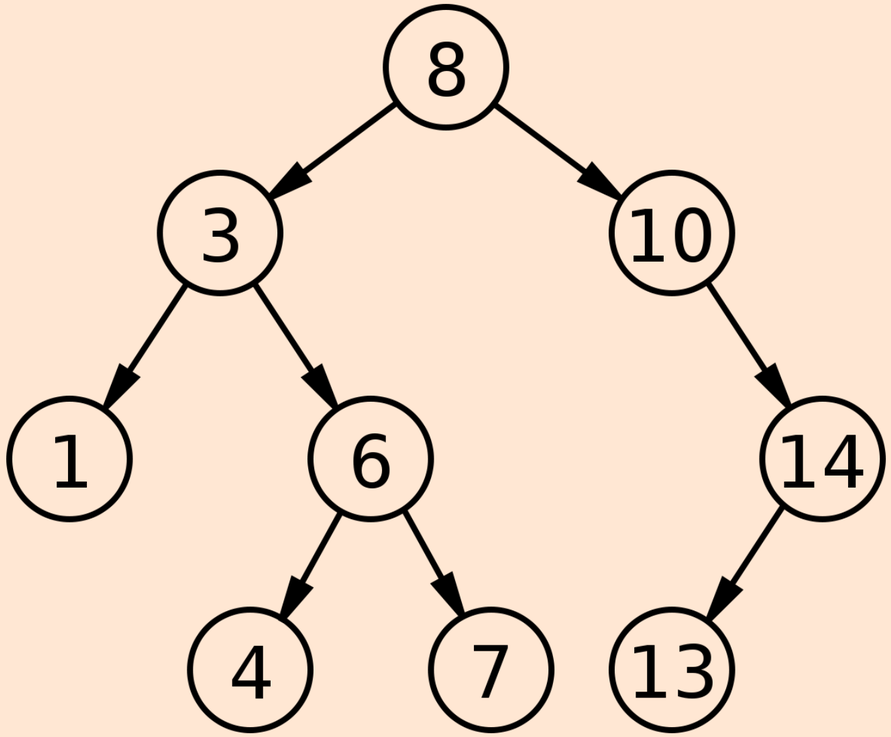
Binary Search Tree

- Ordered Tree - a specific way in which nodes are arranged (based on data in the nodes)
- The left subtree of a node contains only nodes with values lesser than the node’s value.
- The right subtree of a node contains only nodes with values greater than the node’s value.
- The left and right subtree each must also be a binary search tree.



Search, Insert, Delete in a BST

- Let's understand through a **Visualization**!



Let's Code

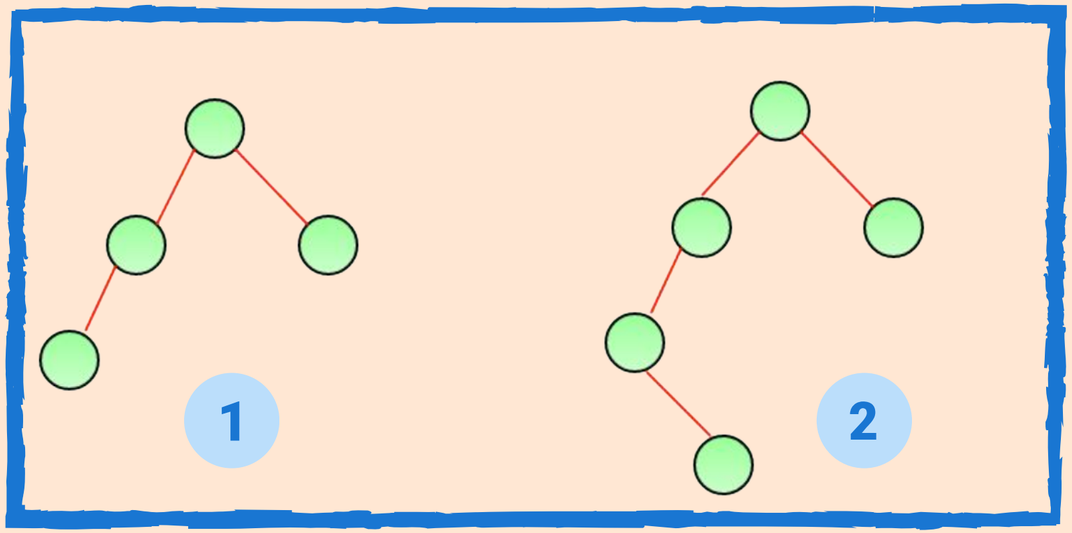
- **Search node in a BST**
- **Insert node in a BST**
- **Delete node in a BST**

Balanced Binary Tree

An empty tree is height-balanced

A non-empty binary tree T is balanced if:
1) Left subtree of T is balanced
2) Right subtree of T is balanced
3) The difference between heights of left subtree and right subtree is not more than 1.

Height-balancing scheme is used in AVL trees.

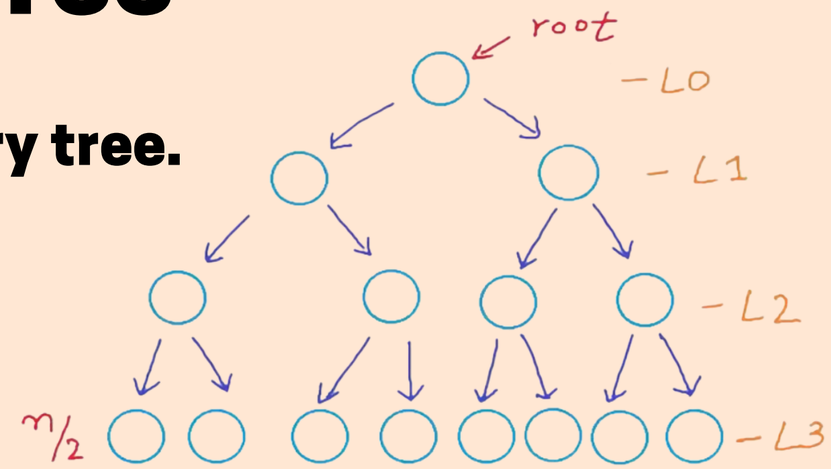


The diagram shows two trees, 1 is height-balanced and 2 is not. The second tree is not height-balanced because height of left subtree is 2 more than height of right subtree.

Height of a balanced Binary Tree

H = O(logn) - most optimal height possible for a binary tree.
where n - number of nodes in the tree

(You can check mathematical proof for this)



Application of this in a BST

Time Complexity of search in a BST = O(H)
In general, O(H) ~ O(n)

But for a balanced BST => O(H) = O(logn)

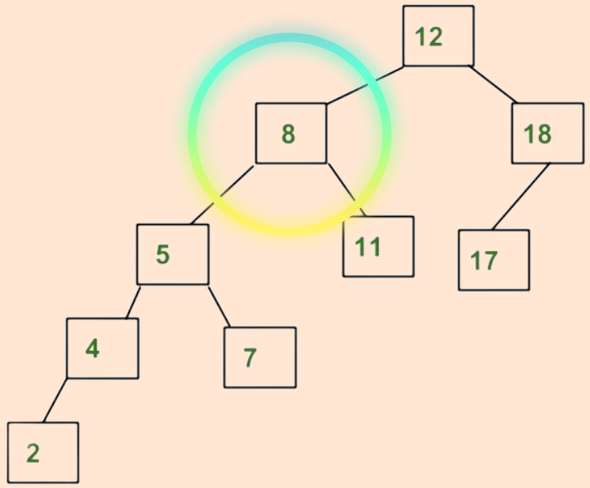
Search, Insertion, Deletion ALL in O(logn) T.C.

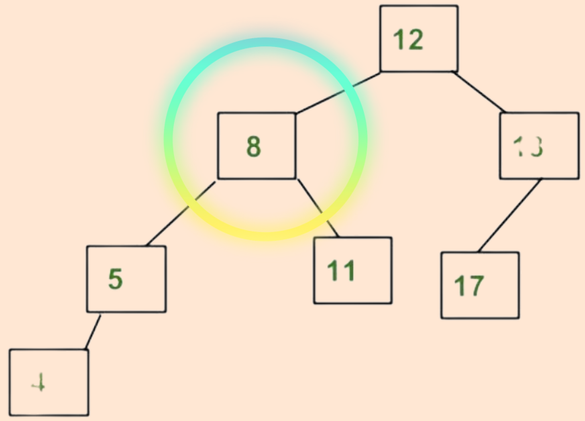
Check if binary tree is balanced

https://leetcode.com/problems/balanced-binary-tree/

AVL Trees

- AVL tree is a **self-balancing Binary Search Tree** (**BST**) **i.e.** difference between heights of left and right subtrees cannot be more than one for all nodes.
- Every language provides some implementation of AVL trees to use directly.
- Let's have a brief **look** at how this self balancing works!





The diagram shows two trees, 1 is not AVL and 2 is AVL.
The second tree is height-balanced whereas 1 is not height balanced because the difference between heights of left and right subtrees for 8 and 12 is greater than 1.

Construct Binary Tree from Preorder and Inorder Traversal

https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/