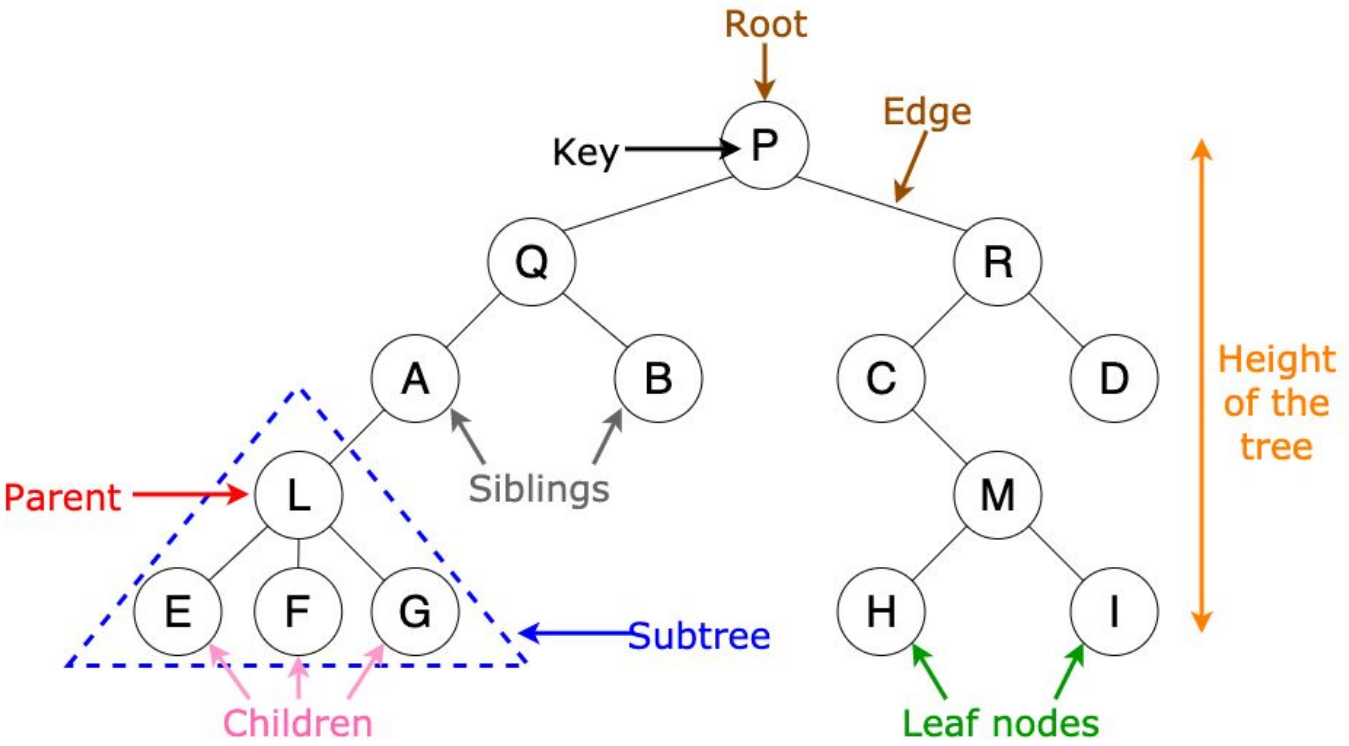
Data Structures

Lecture 11
Tree

Credit: Prantik Paul, Lecturer, BRAC University

Trees


Root
Edge
Key
P
Q
R
A
B
C
D
L
M
E
F
G
H
I
Height of the tree
Parent
Siblings
Subtree
Children
Leaf nodes

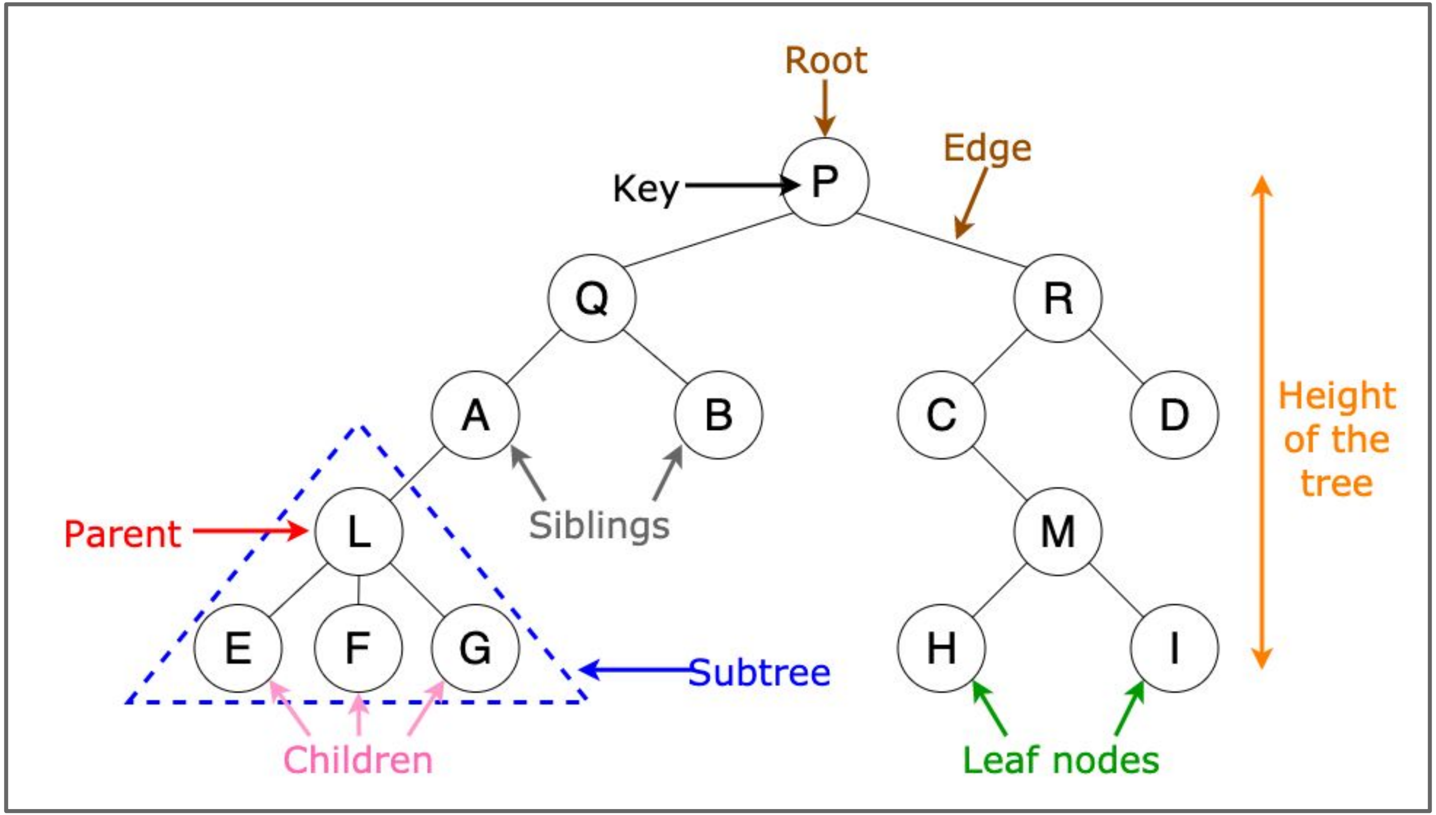
Why Trees?

Sorting New Elements

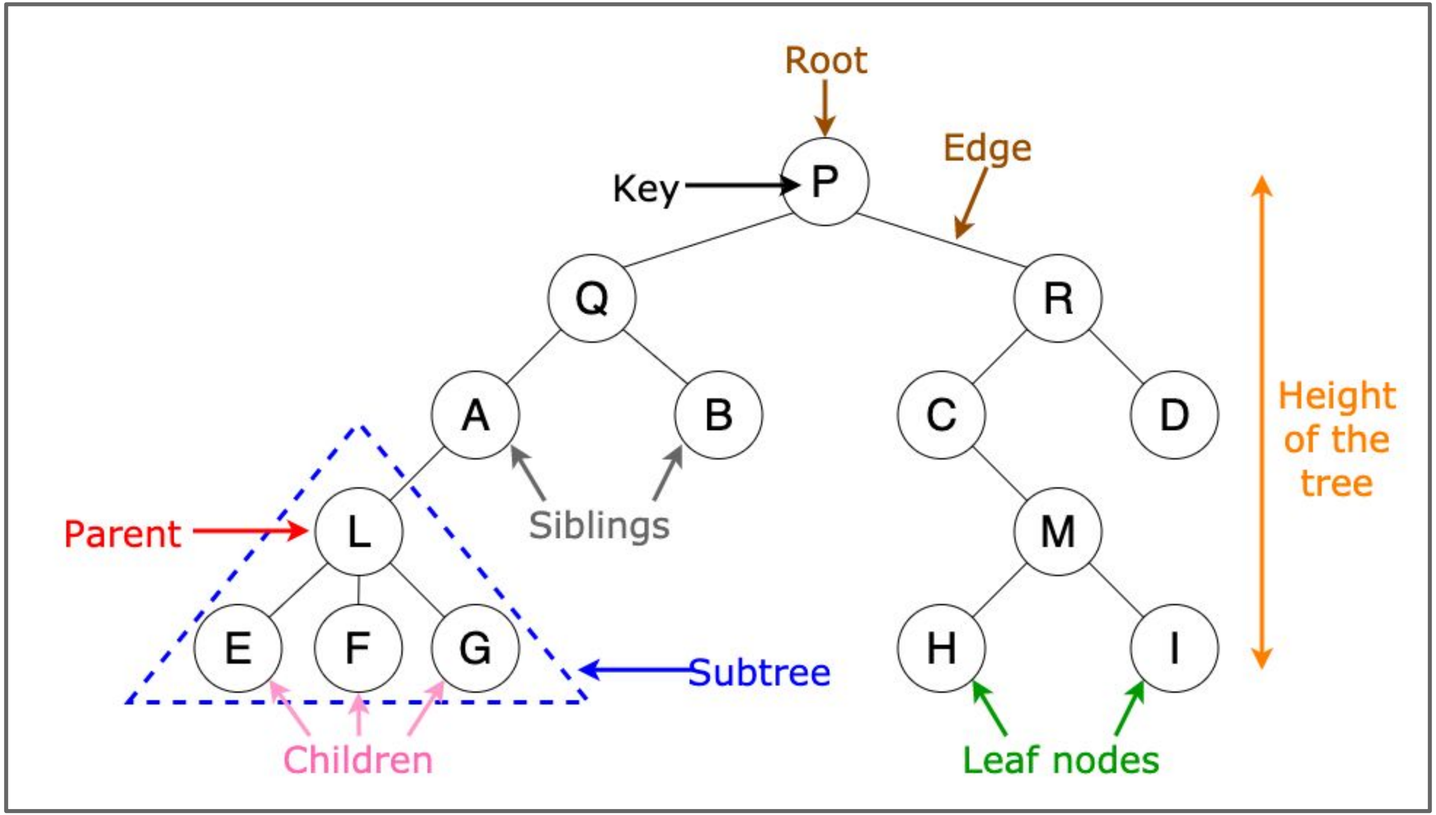
Folder/File System Structure

Computer Network Algorithms

Trees - Root/Leaf/Non-Leaf

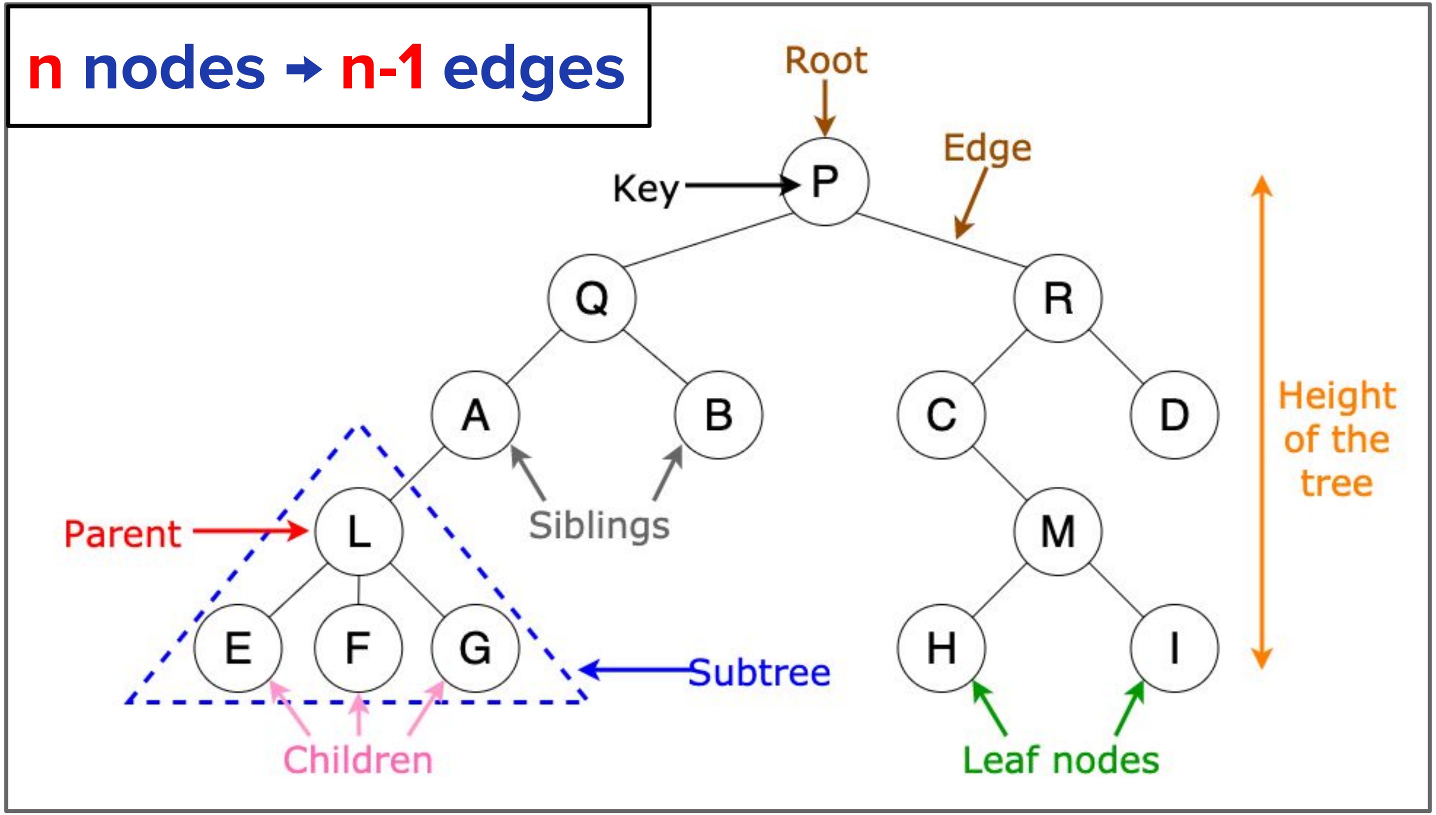


Trees - Parent/Child/Siblings

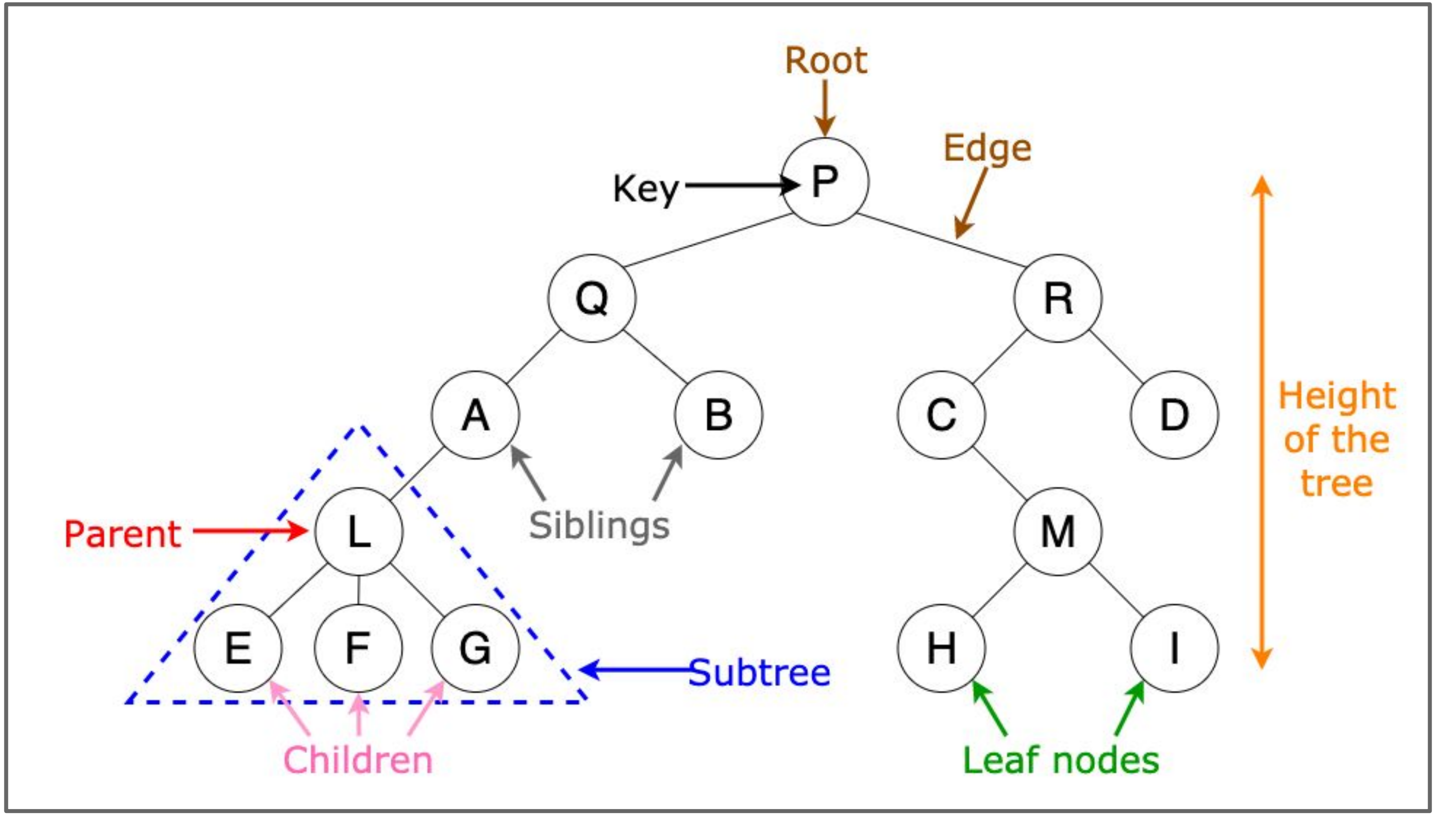


Trees - Edge/Path

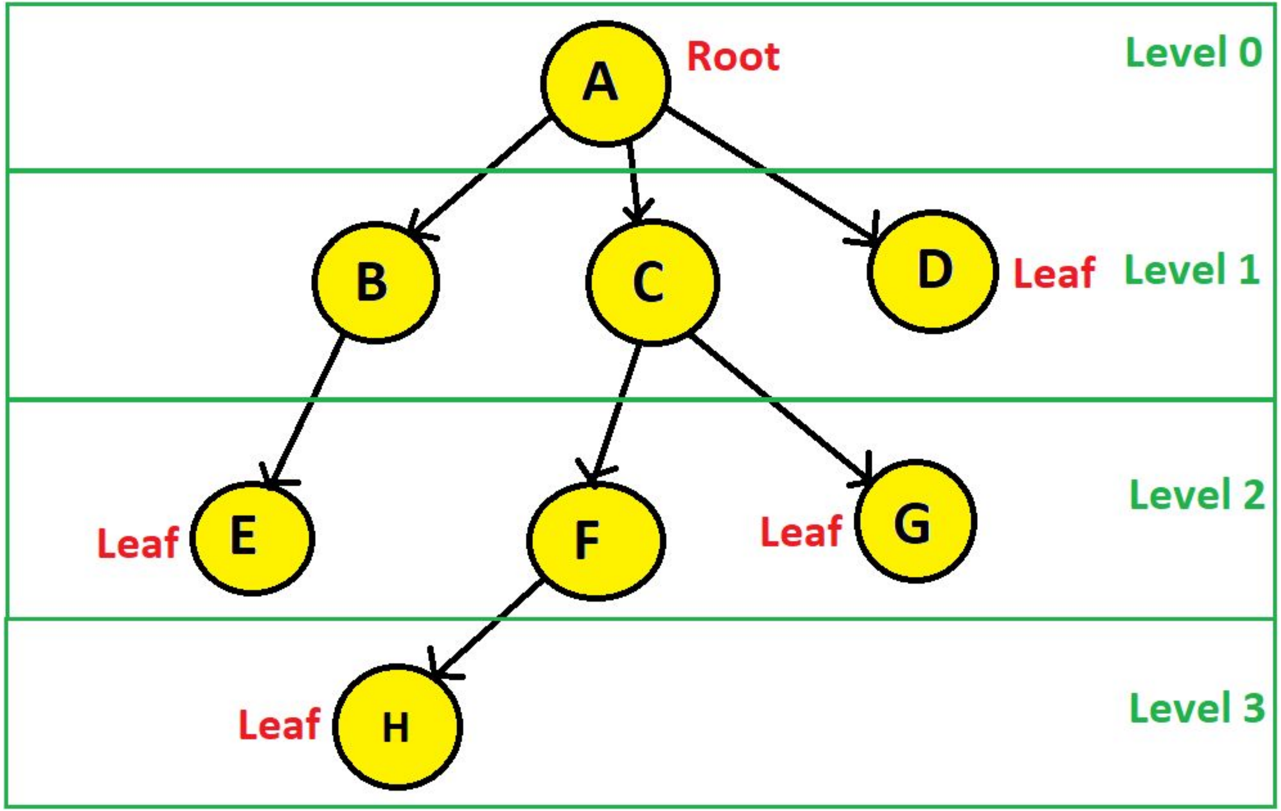
n nodes → n-1 edges



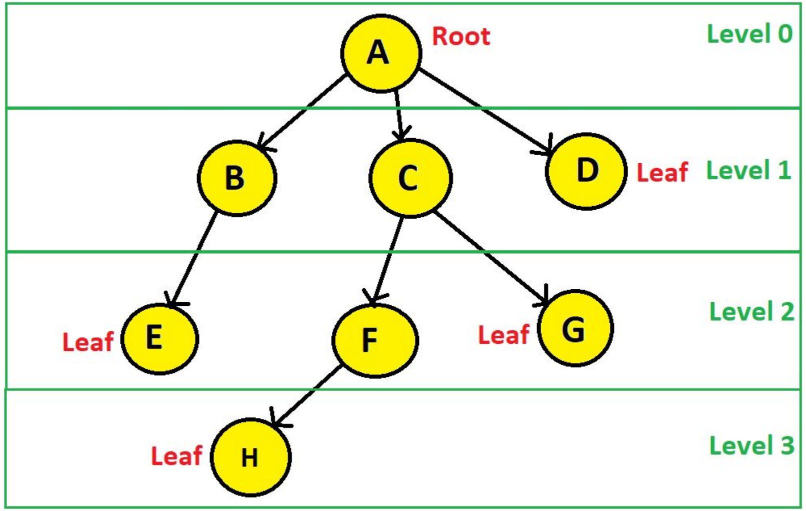
Trees - Degree



Trees - Depth/Height/Level



Trees - Depth/Height/Level



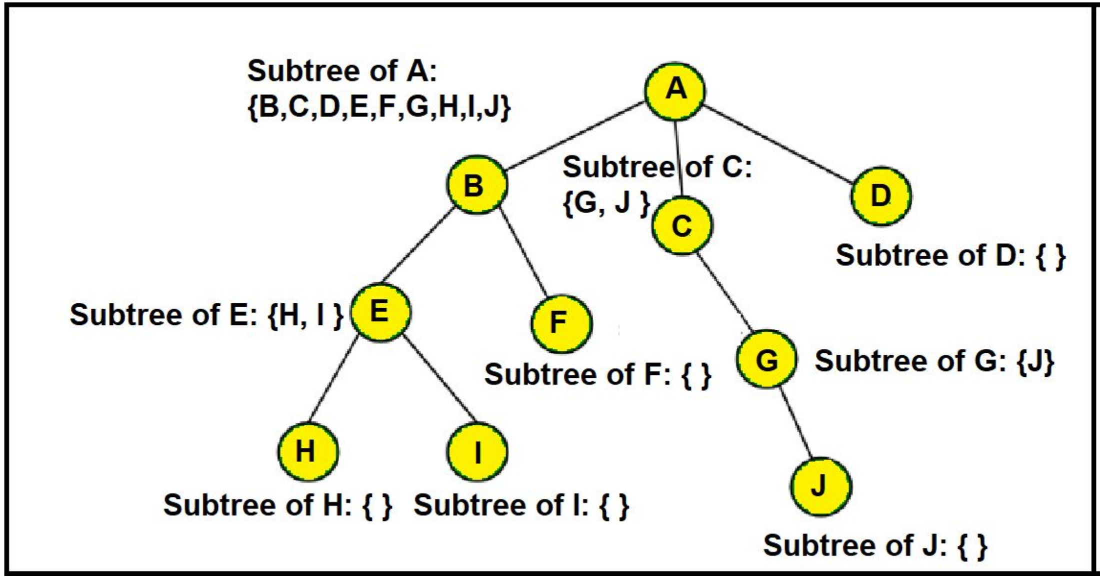
Depth of A?

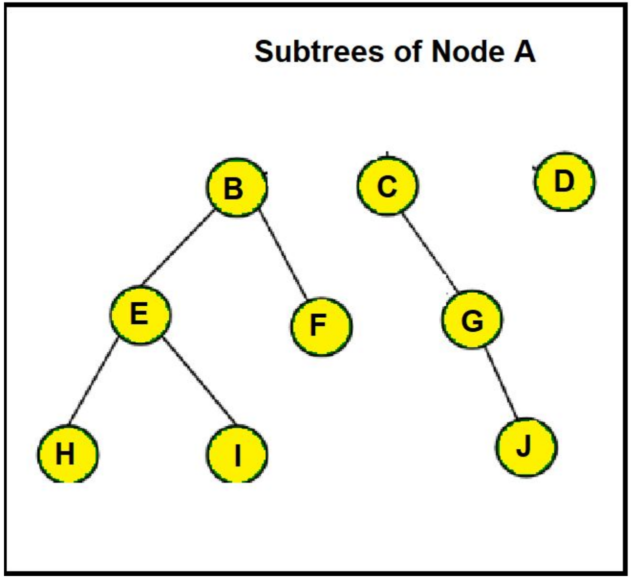
Height of A?

Depth != Height

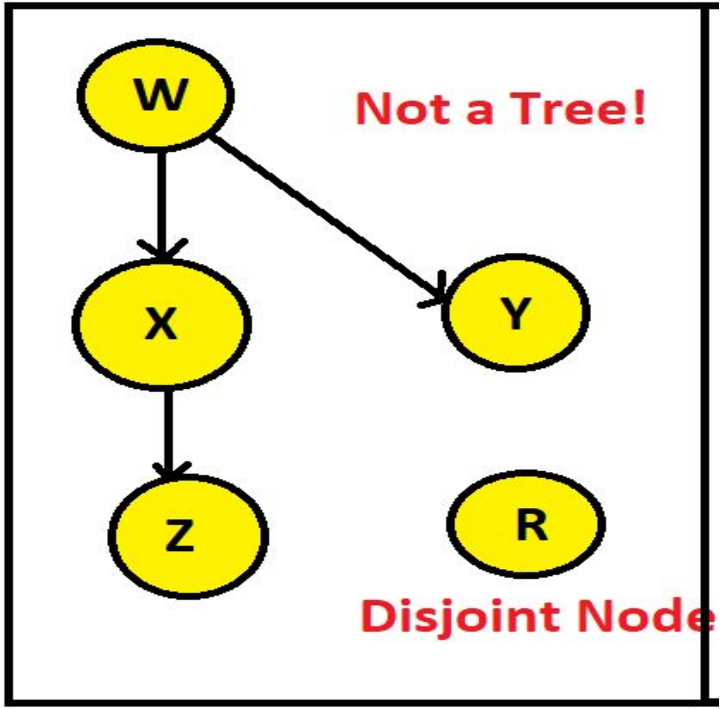
Depth = Level

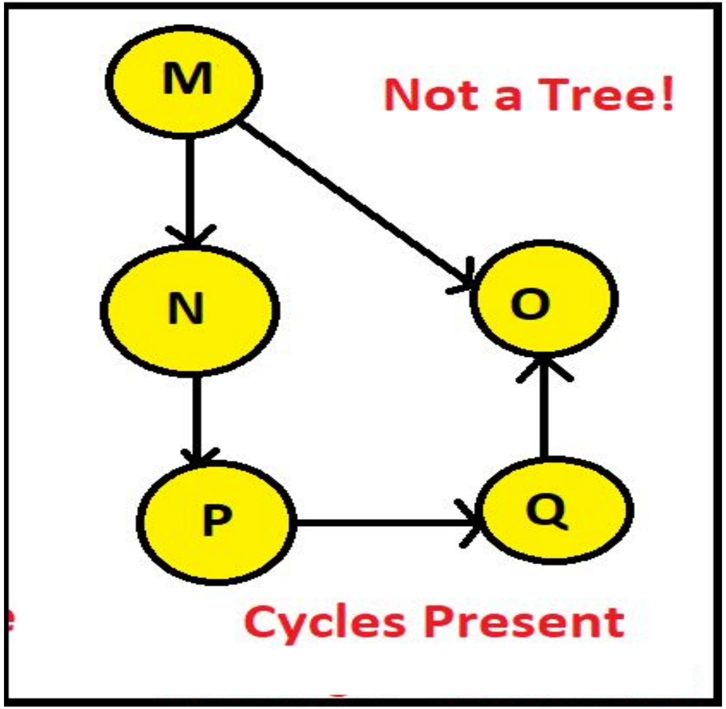
Trees - Subtree



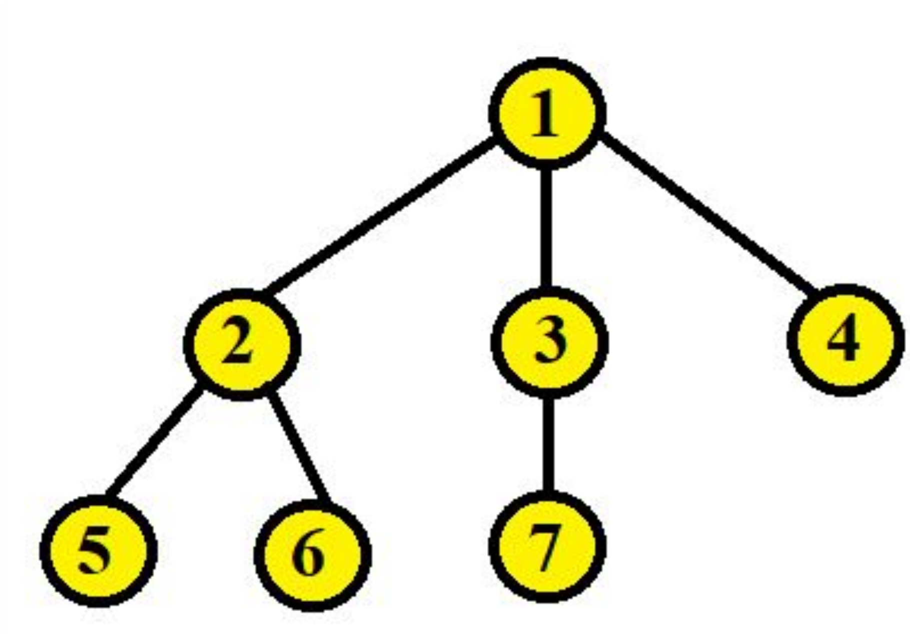


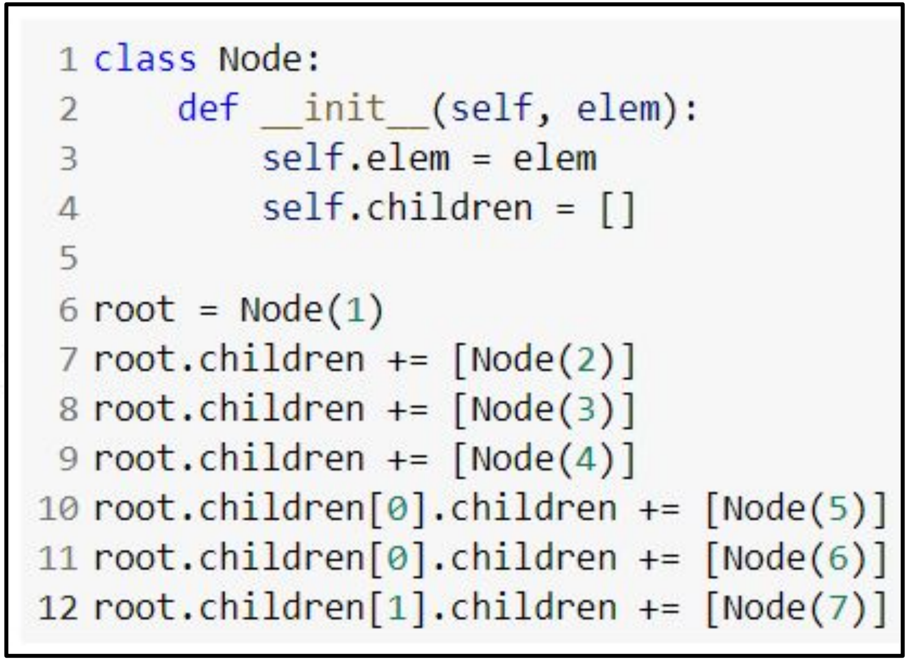
Trees - Characteristics





Trees - Build a Tree



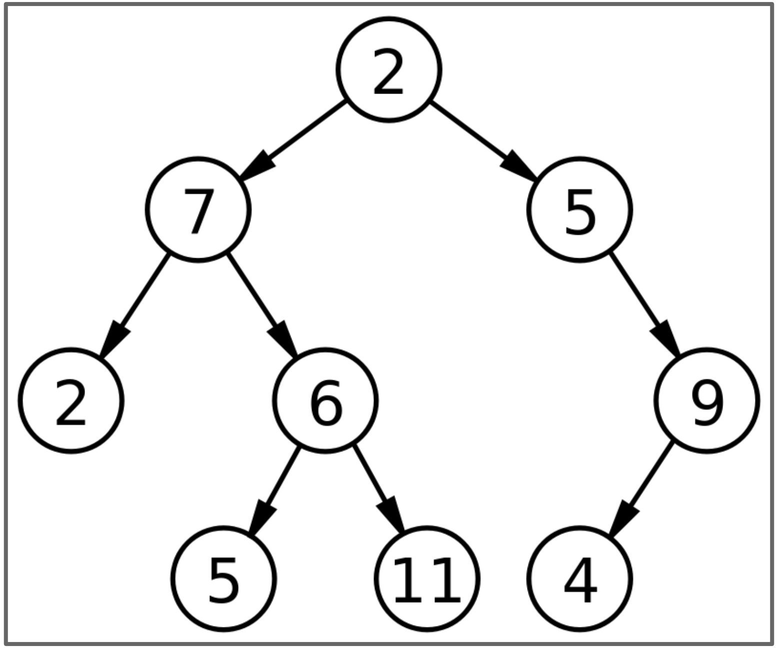


```
class Node:
    def __init__(self, elem):
        self.elem = elem
        self.children = []

root = Node(1)
root.children += [Node(2)]
root.children += [Node(3)]
root.children += [Node(4)]
root.children[0].children += [Node(5)]
root.children[0].children += [Node(6)]
root.children[1].children += [Node(7)]
```

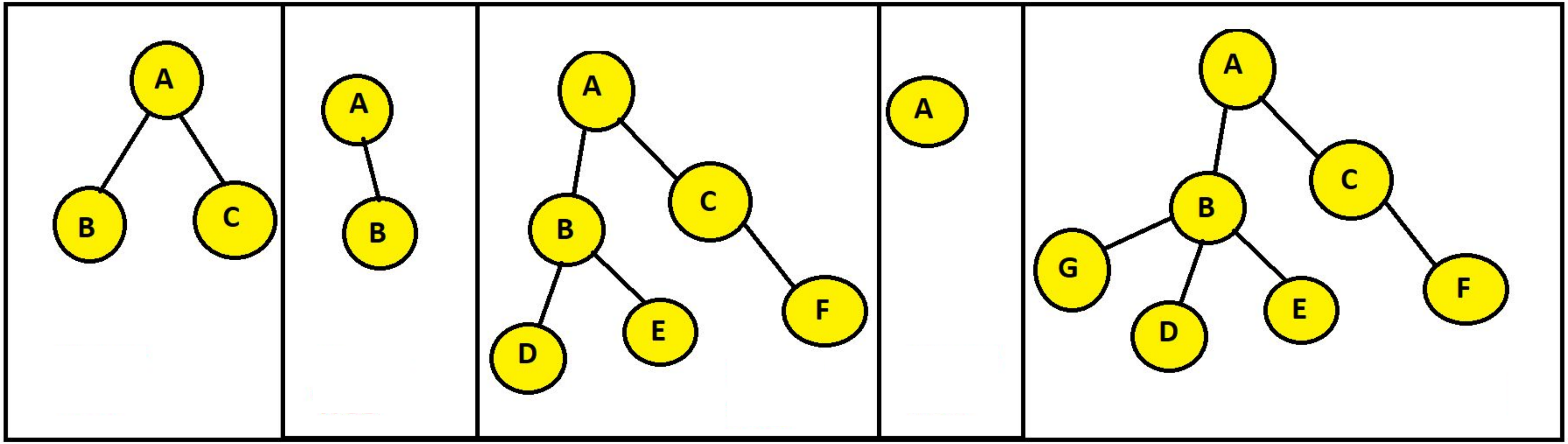
Dynamic Representation with Linked List

Sequential Representation with Array

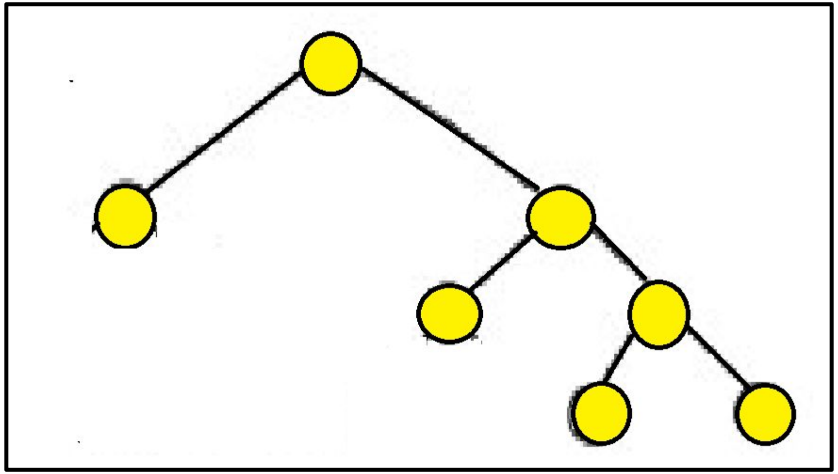
Binary Trees

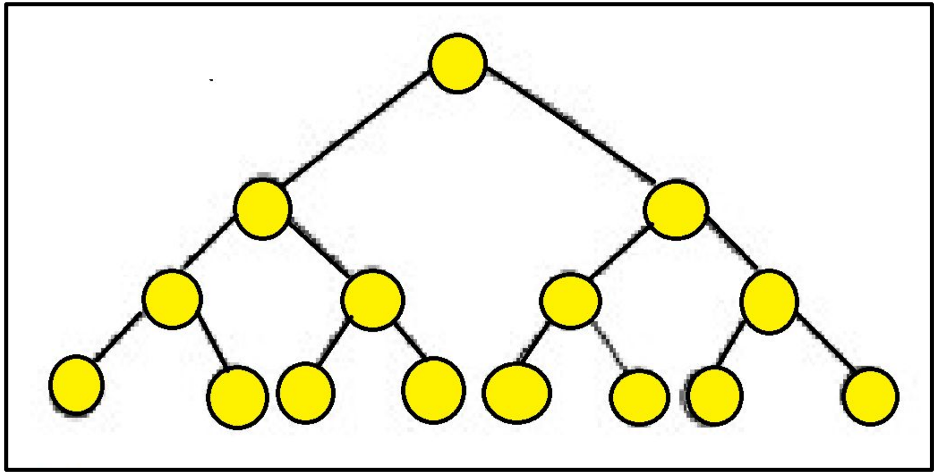


Binary Trees



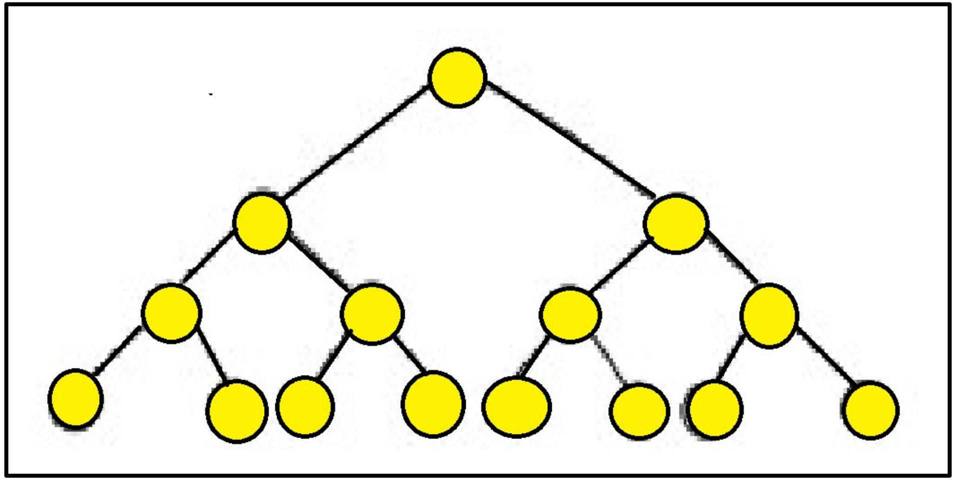
Binary Tree - Full/Strict Binary Tree

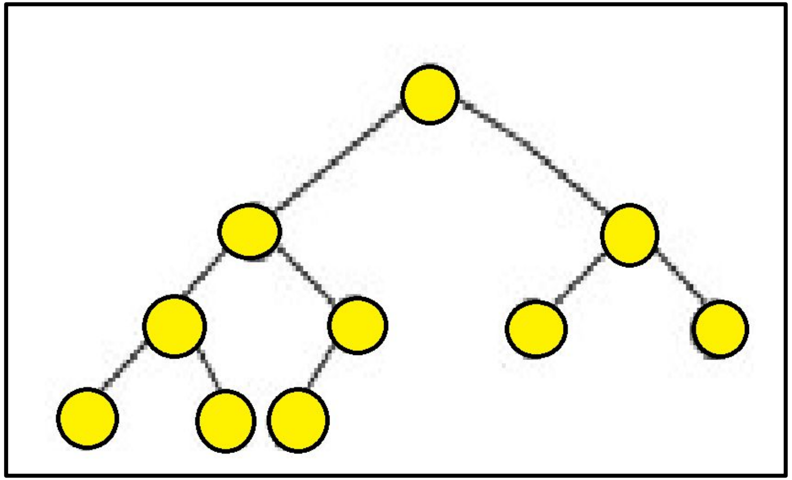




No of leaf nodes = no of internal nodes + 1

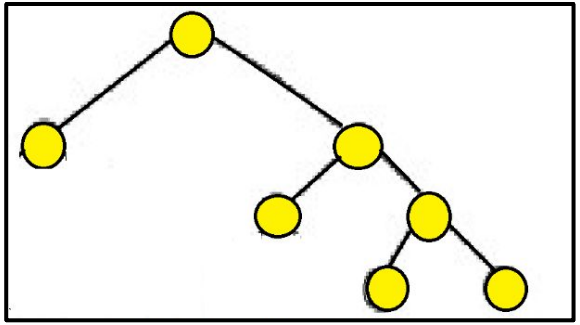
Binary Tree - Complete Binary Tree

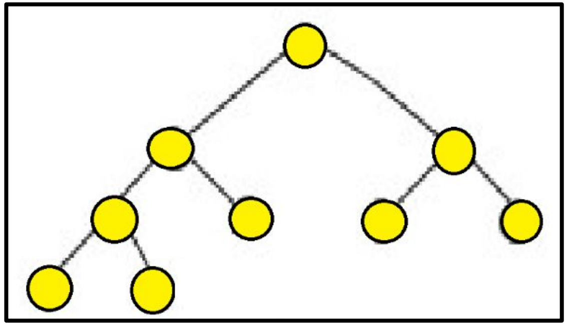


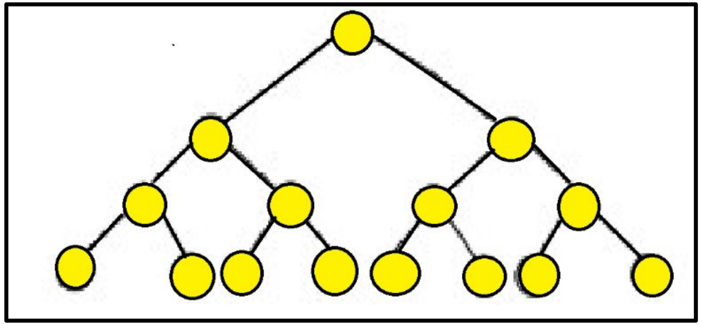


All levels filled starting from LEFT

Binary Tree - Perfect Binary Tree



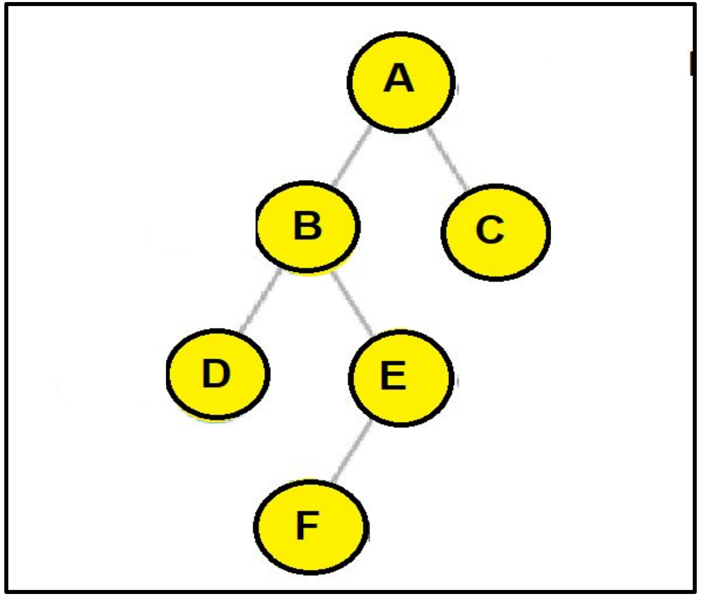


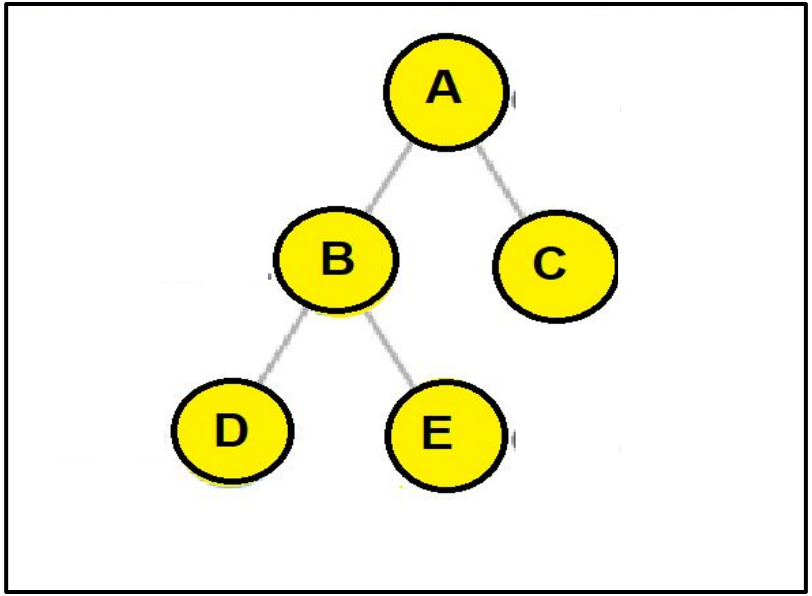


All Internal Nodes have 2 child

All Leaf Nodes at same Level

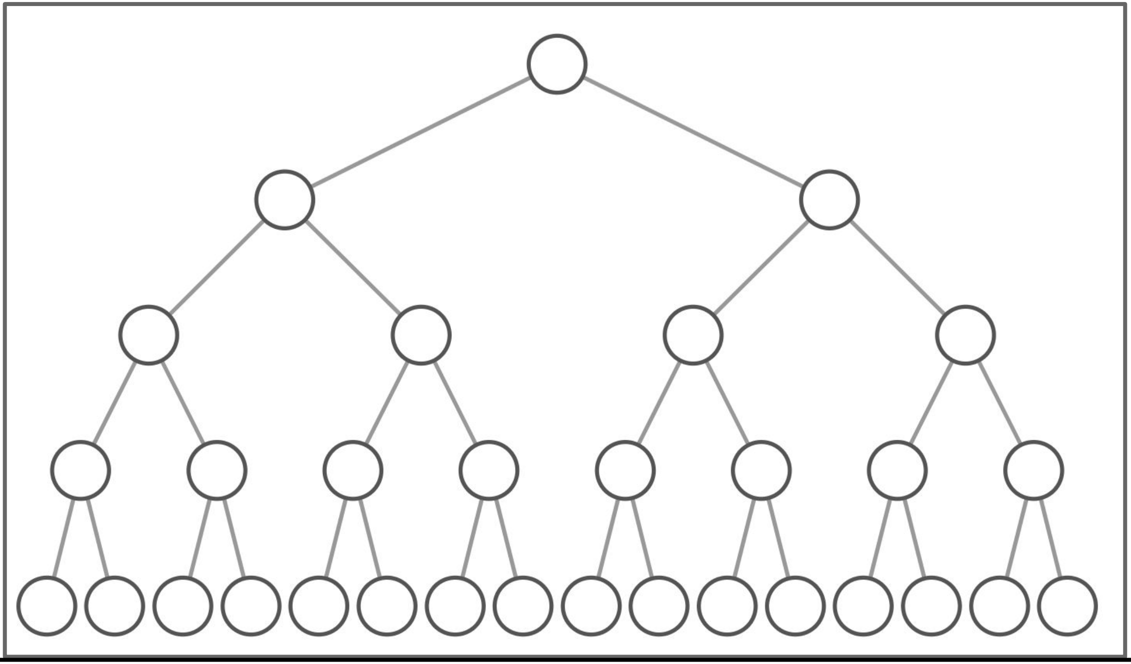
Binary Tree - Balanced Binary Tree





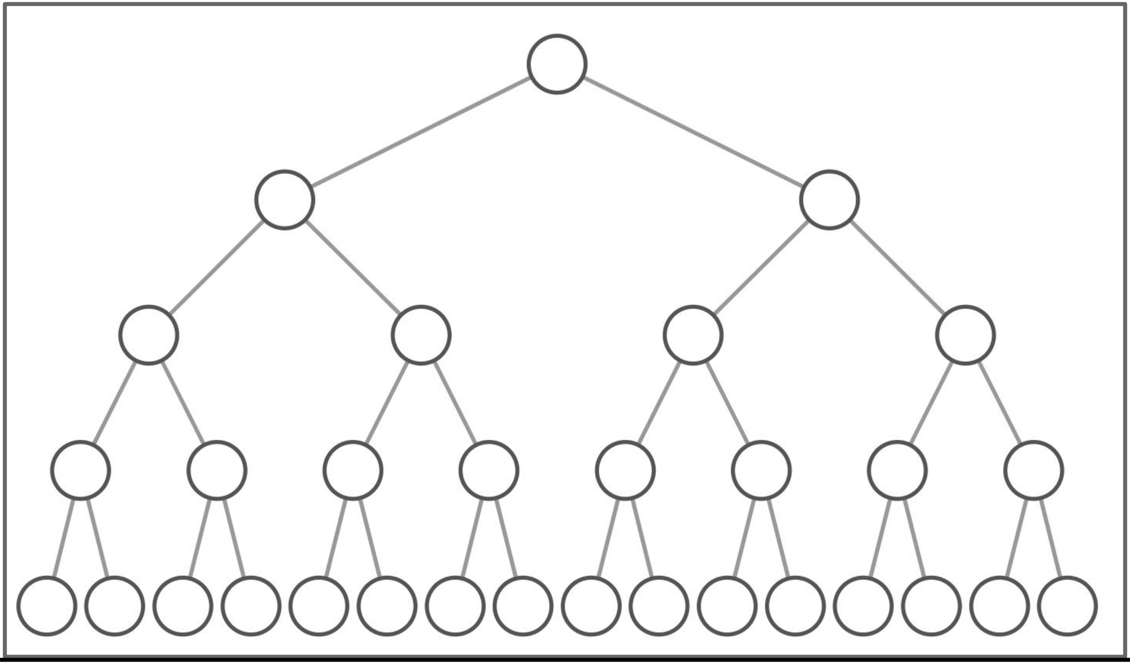
$$| \text{height(left)} - \text{height(right)} | <= 1$$

Binary Trees - Characteristics



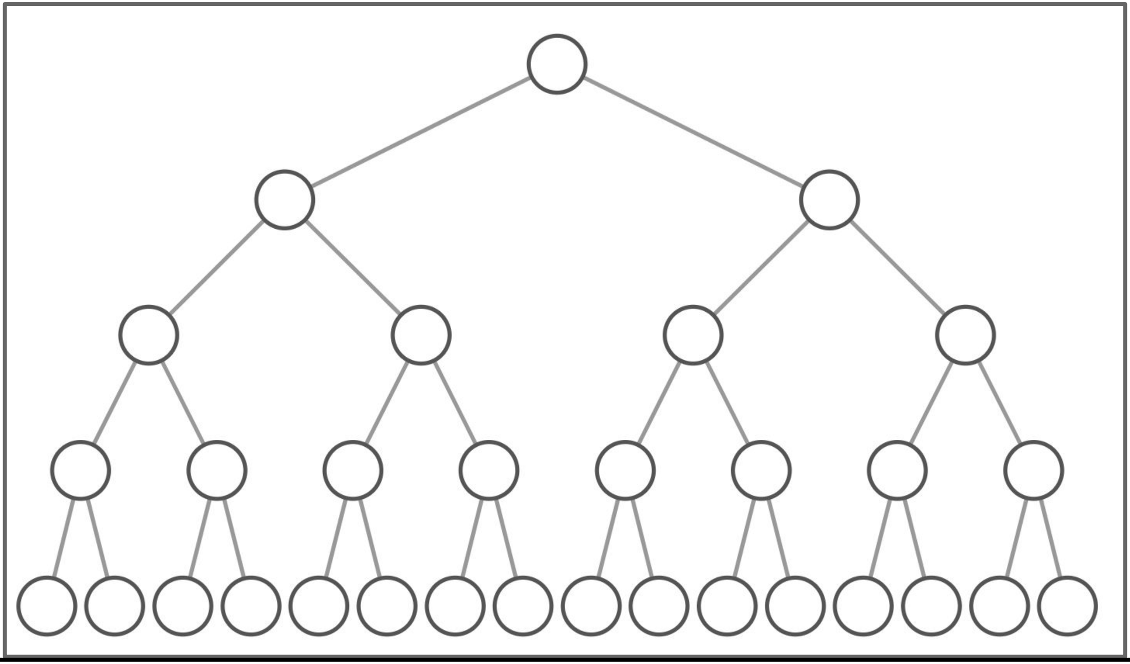
The maximum number of nodes at level i is:

Binary Trees - Characteristics



The maximum number of nodes possible in a binary tree of height ‘h’ is:

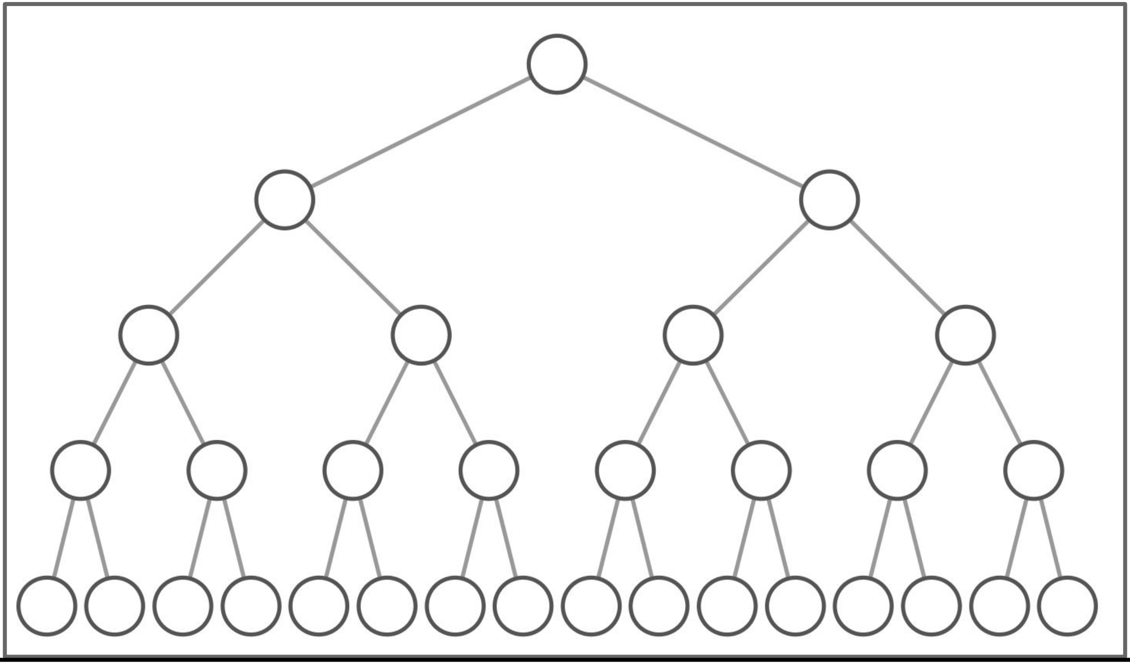
Binary Trees - Characteristics



Number of internal nodes : n

Number of external nodes :

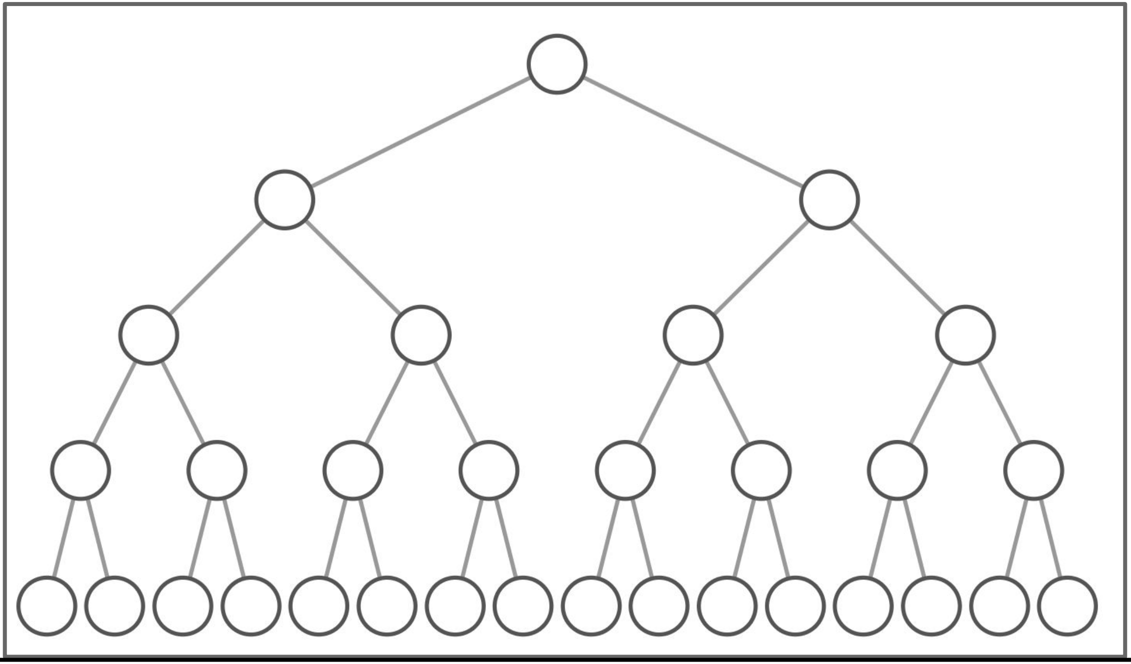
Binary Trees - Characteristics



Number of internal nodes : n

Number of internal edges :

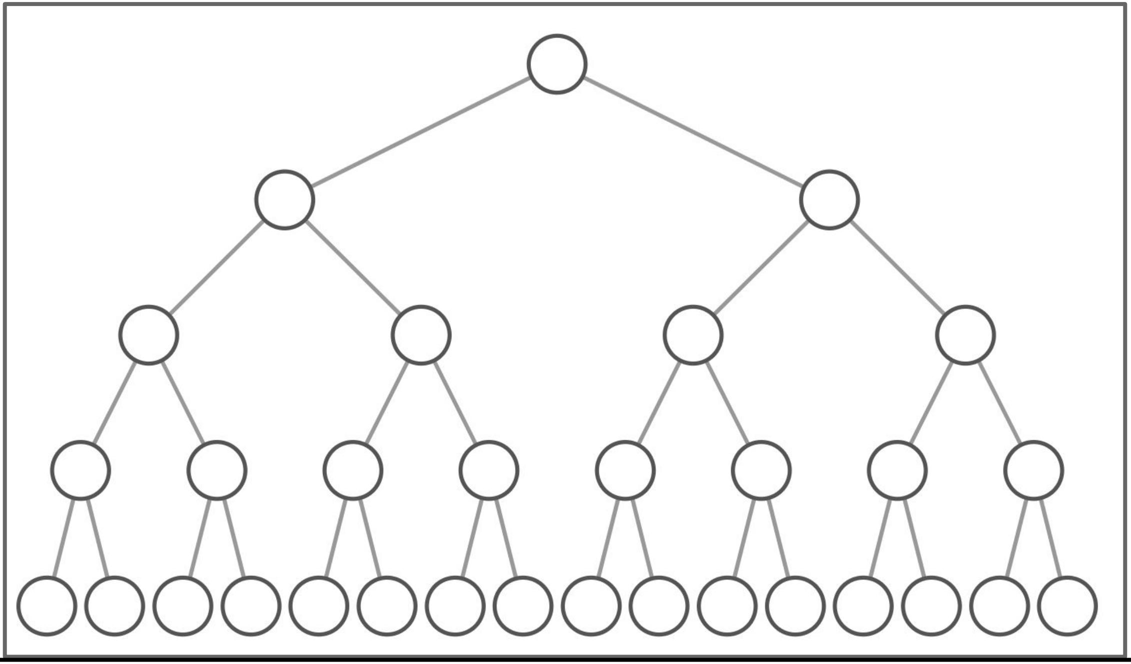
Binary Trees - Characteristics



Number of internal nodes : n

Number of external edges :

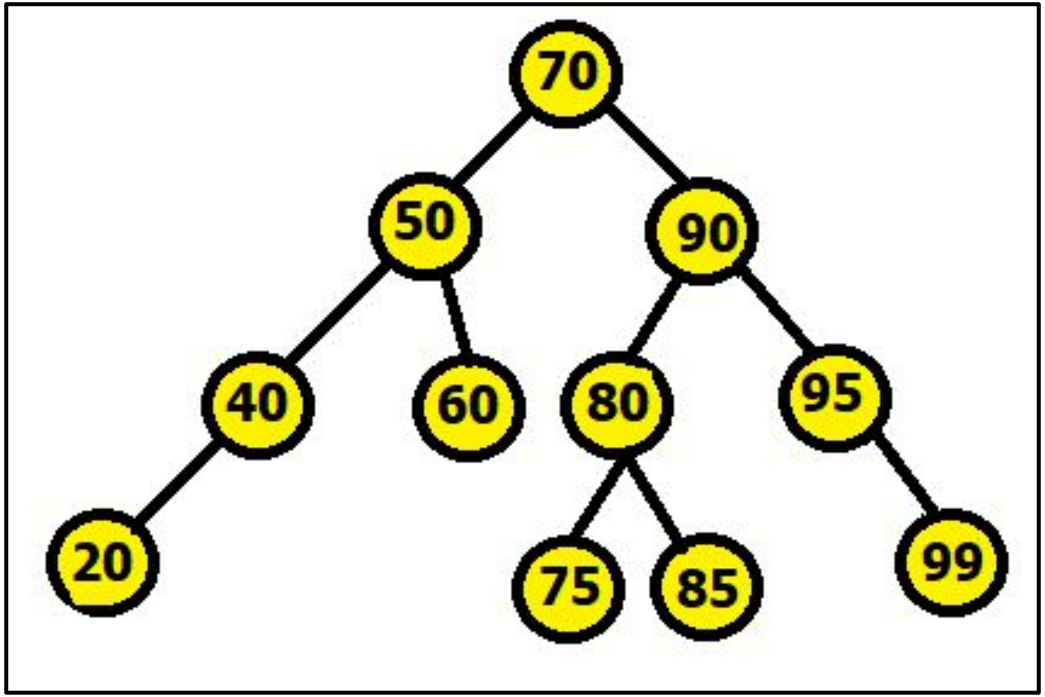
Binary Trees - Characteristics



Number of internal nodes : n

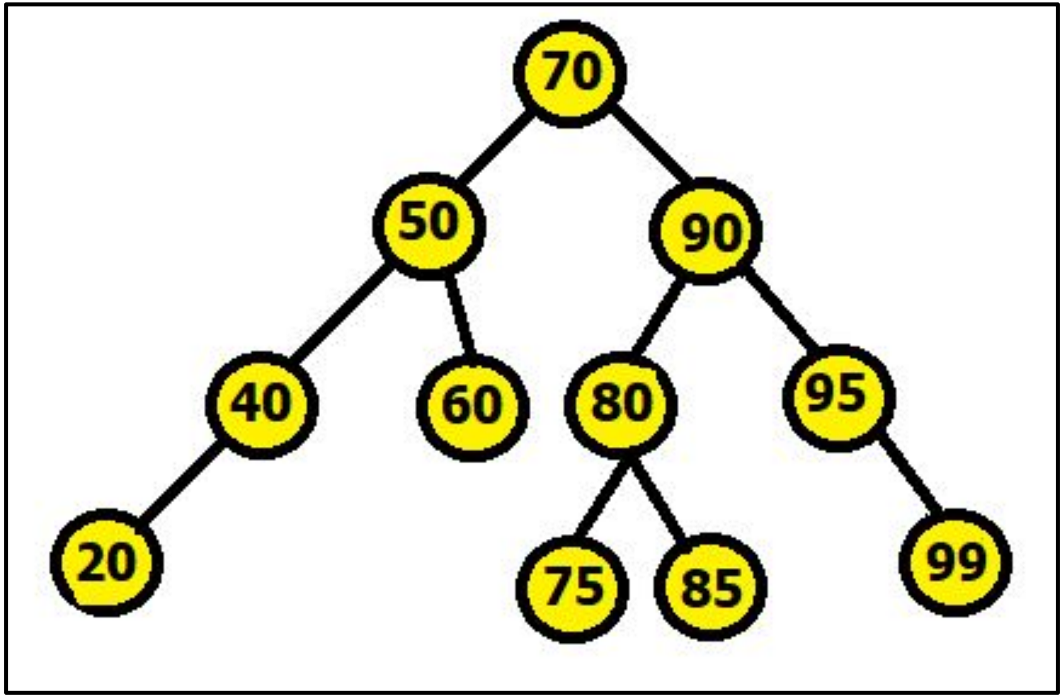
Number of edges :

Binary Trees - Traversal (Pre-order)



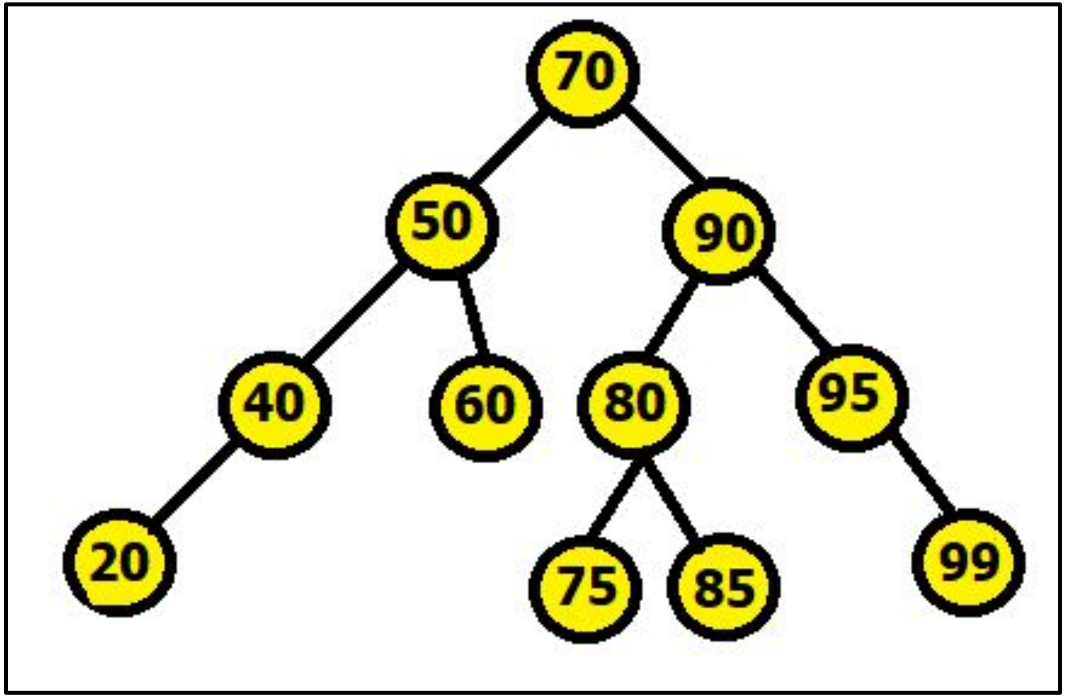
Pre-Order: 70, 50, 40, 20, 60, 90, 80, 75, 85, 95, 99

Binary Trees - Traversal (In-order)



In-Order: 20, 40, 50, 60, 70, 75, 80, 85, 90, 95, 99

Binary Trees - Traversal (Post-order)



Post-Order: 20, 40, 60, 50, 75, 85, 80, 99, 95, 90, 70

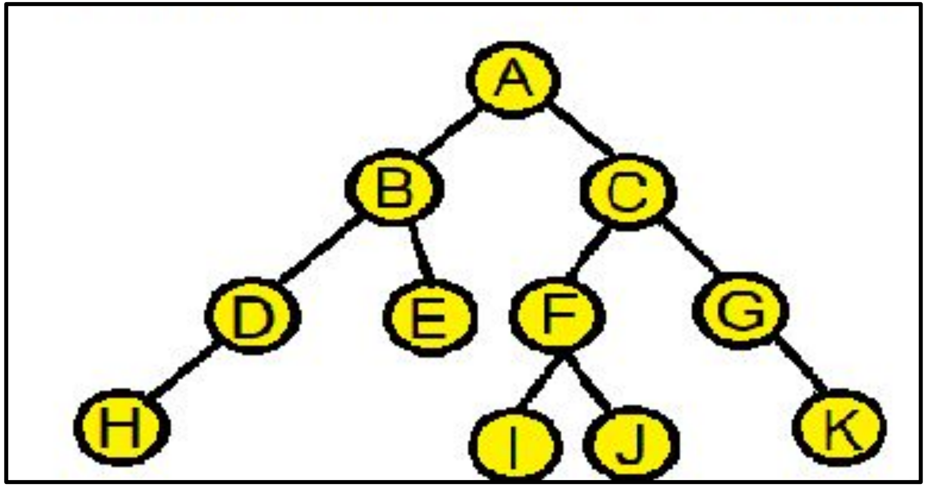
Binary Trees - Array Representation

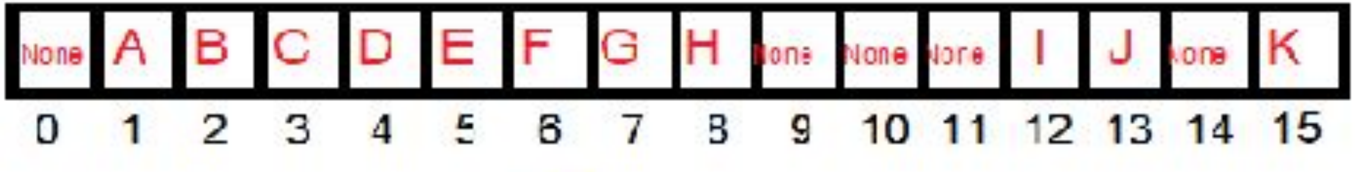
If the **height** of the binary tree if **h**, An array of maximum 2^{h+1} **length** is required

The **root** is placed at **index 1**

Any node that is placed at index i, will have its **left child** placed at **2i** and its **right child** at **2i+1**

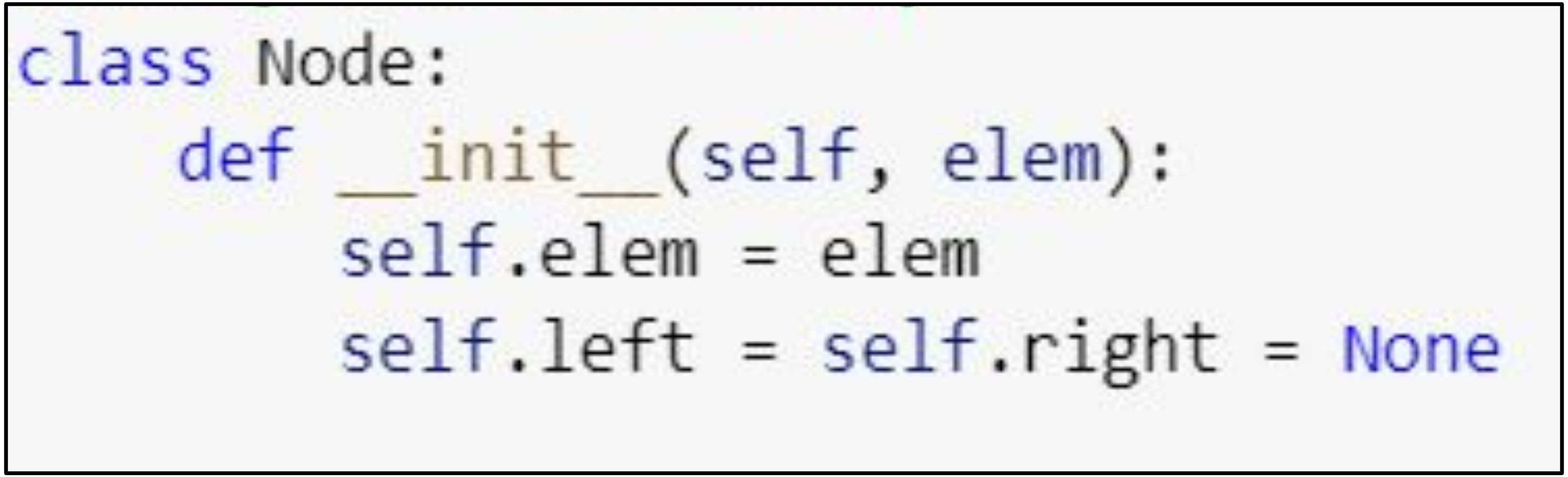
Binary Trees - Array Representation





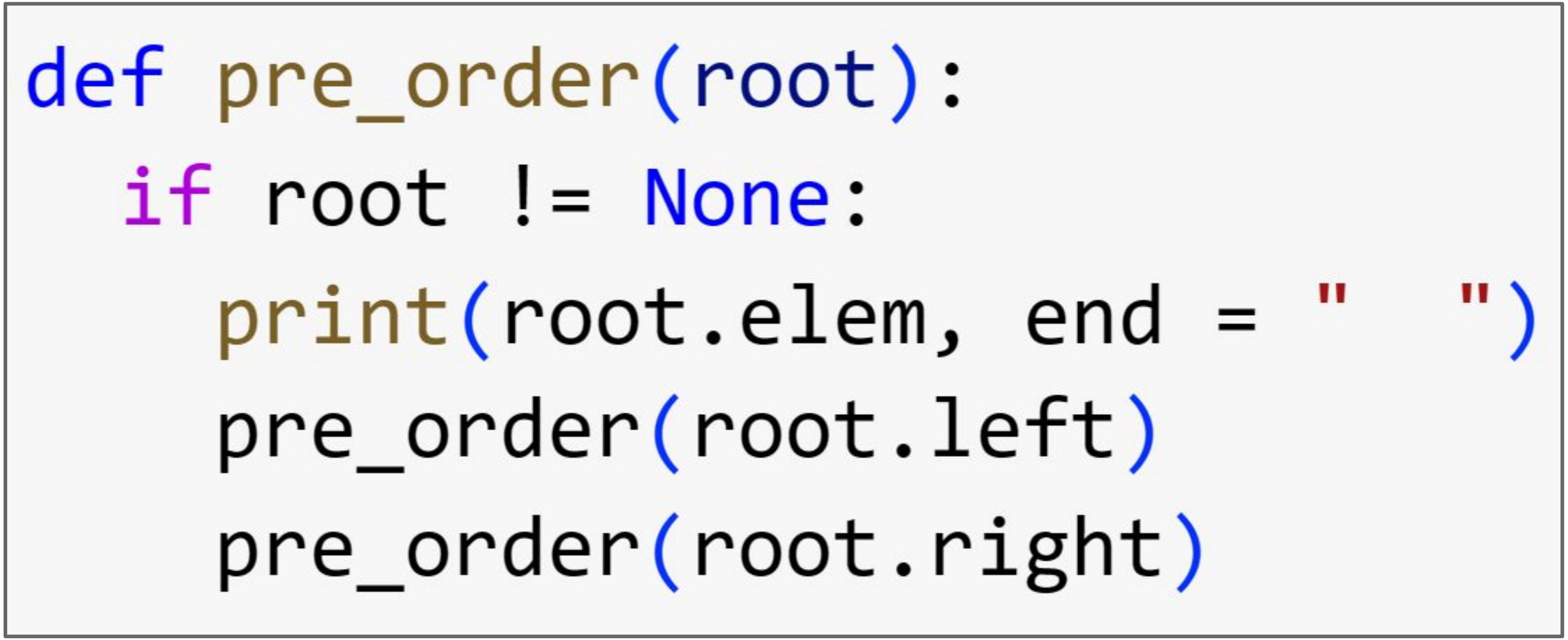
None	A	B	C	D	E	F	G	H	None	None	None	I	J	None	K
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Binary Trees - Tree Node



```
class Node:
    def __init__(self, elem):
        self.elem = elem
        self.left = self.right = None
```

Binary Trees - Preorder Traversal



```
def pre_order(root):
  if root != None:
    print(root.elem, end = "  ")
    pre_order(root.left)
    pre_order(root.right)
```

Binary Trees - Inorder Traversal

Left → print(root.elem) → Right

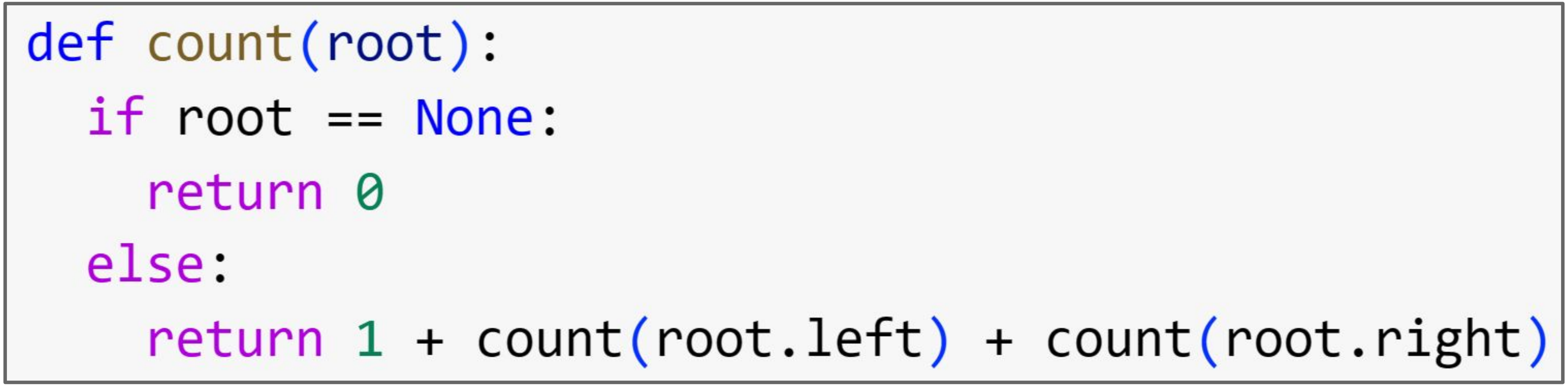
Do yourself

Binary Trees - Postorder Traversal

Left → Right → print(root.elem)

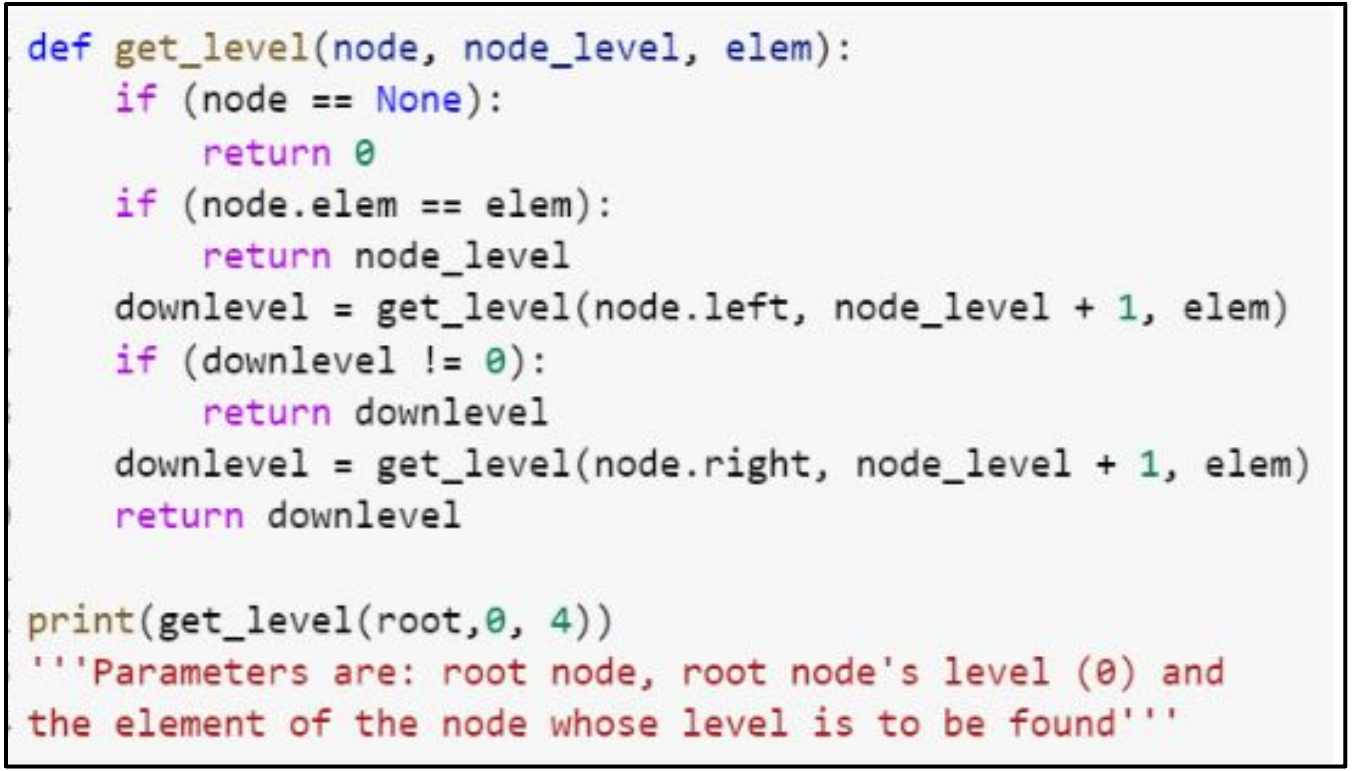
Do yourself

Binary Trees - Count Nodes of Tree



```
def count(root):
  if root == None:
    return 0
  else:
    return 1 + count(root.left) + count(root.right)
```

Binary Trees - Find Level of a Node



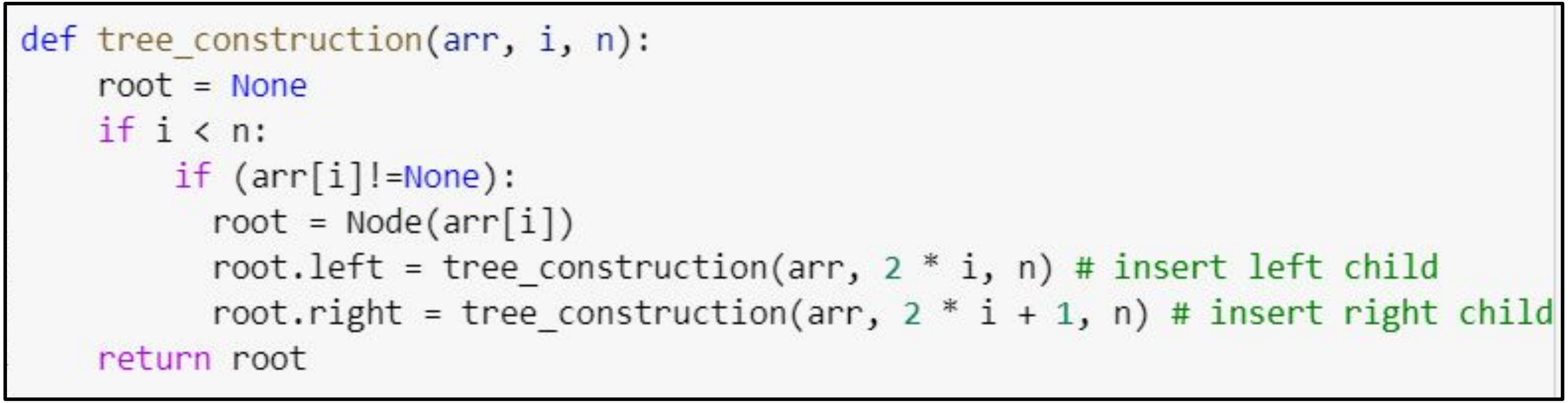
```
def get_level(node, node_level, elem):
    if (node == None):
        return 0
    if (node.elem == elem):
        return node_level
    downlevel = get_level(node.left, node_level + 1, elem)
    if (downlevel != 0):
        return downlevel
    downlevel = get_level(node.right, node_level + 1, elem)
    return downlevel

print(get_level(root,0, 4))
'''Parameters are: root node, root node's level (0) and
the element of the node whose level is to be found'''
```

Binary Trees - Find Height of a Node

```
If node is None
    return -1
else
    return 1 + max(left_subtree_height, right_subtree_height)
```

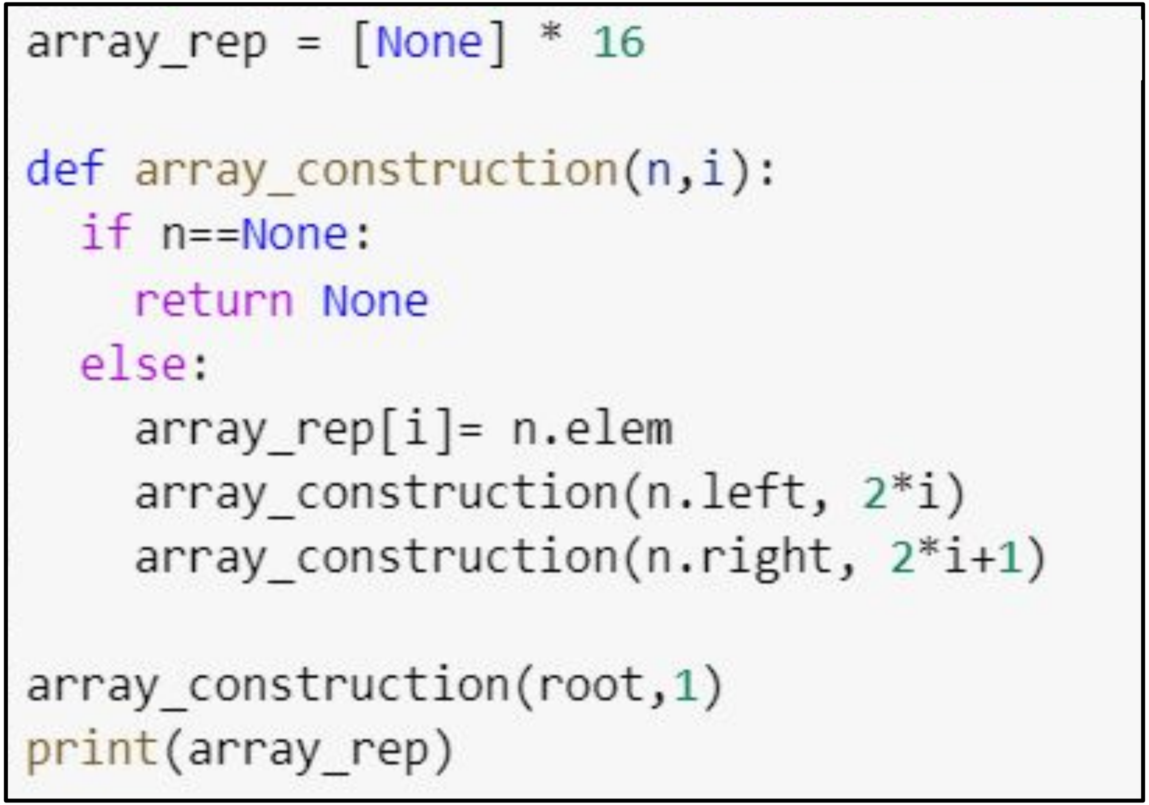
Do yourself

Binary Trees - Build Tree From Array



```
def tree_construction(arr, i, n):
    root = None
    if i < n:
        if (arr[i]!=None):
            root = Node(arr[i])
            root.left = tree_construction(arr, 2 * i, n) # insert left child
            root.right = tree_construction(arr, 2 * i + 1, n) # insert right child
    return root
```

Binary Trees - Build Array From Tree



```
array_rep = [None] * 16

def array_construction(n,i):
  if n==None:
    return None
  else:
    array_rep[i]= n.elem
    array_construction(n.left, 2*i)
    array_construction(n.right, 2*i+1)

array_construction(root,1)
print(array_rep)
```