Data Structure - Spring 2022

11. Tree

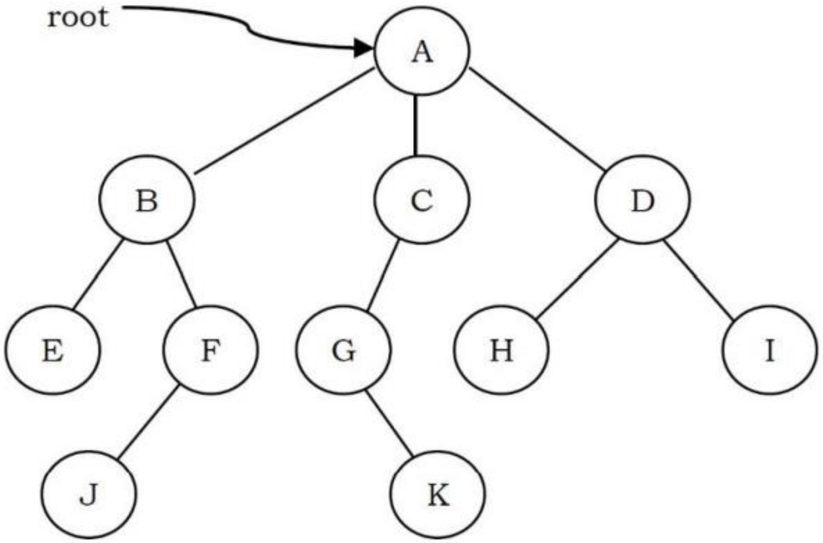
Walid Abdullah Al
Computer and Electronic Systems Engineering
Hankuk University of Foreign Studies

Based on:
Goodrich, Chapter 8
Karumanchi, Chapter 6
Slides by Prof. Yung Yi, KAIST
Slides by Prof. Chansu Shin, HUFS

Computer Vision Lab
Hankuk University of Foreign Studies

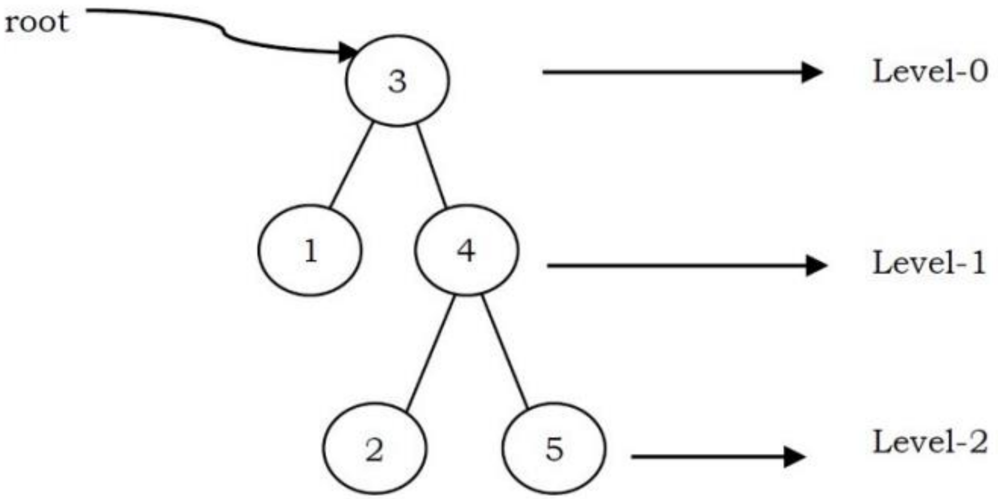
Tree

- Similar to linked list
- But each node can point to multiple nodes



- Root: no parents
- Edge: link from parent to child
- Leaf node: no children
- Siblings: children of the same parent

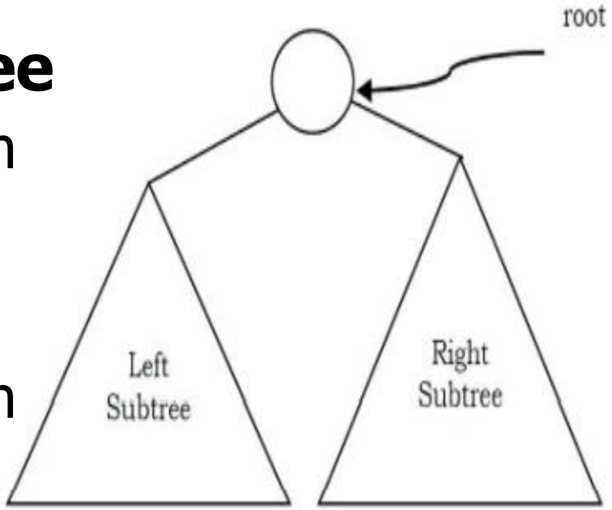
Level, depth and height

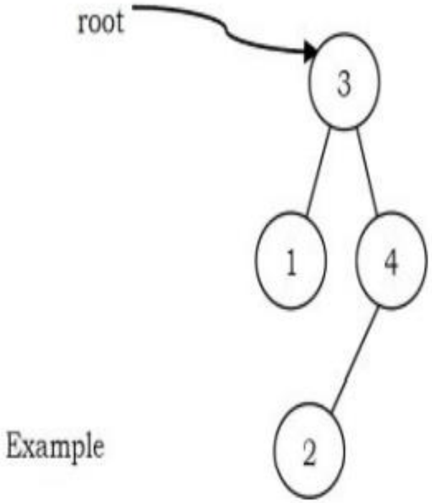


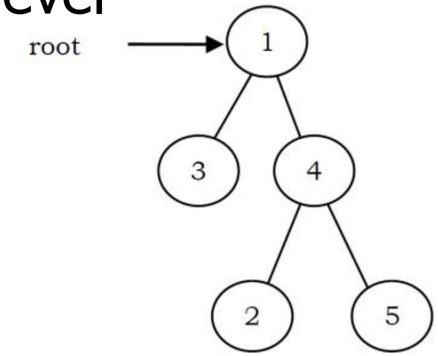
- **Depth of a node:** path-length from the root
- **Height of a tree:** path-length from the root to the deepest node
- **Size:** total number of nodes in a tree

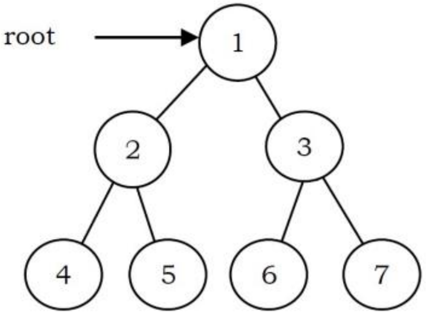
Binary Tree

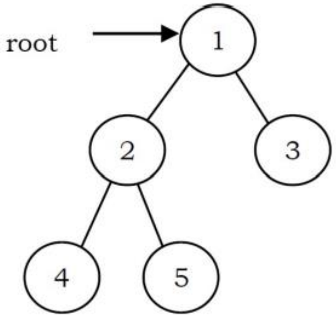
- Each node has ≤ 2 children
- **Proper binary tree**
  - Exactly 2 children
  - Or, no children
- **Full binary tree**
  - Exactly 2 children
  - and,
  - all leaf nodes
    are at same level



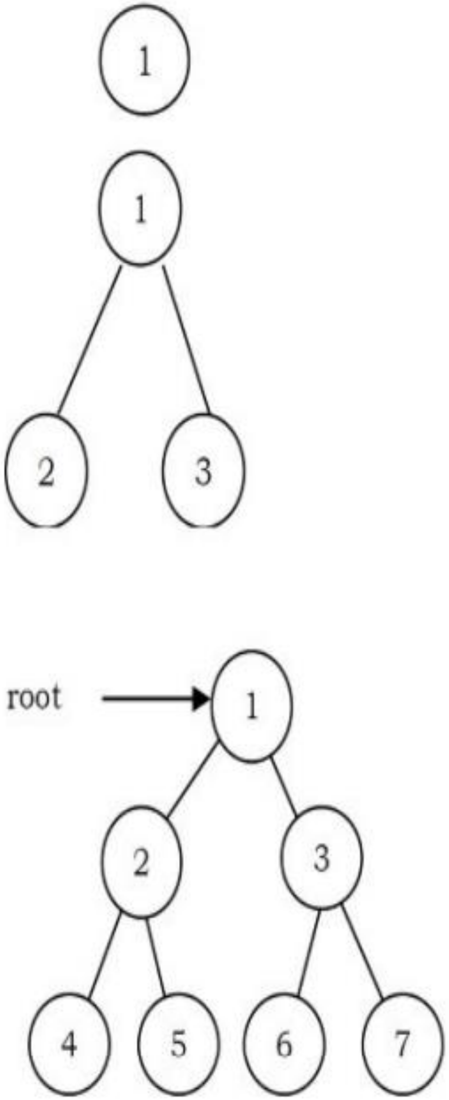








Height vs size

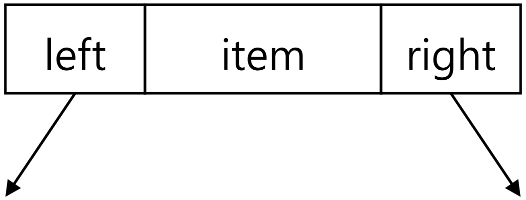


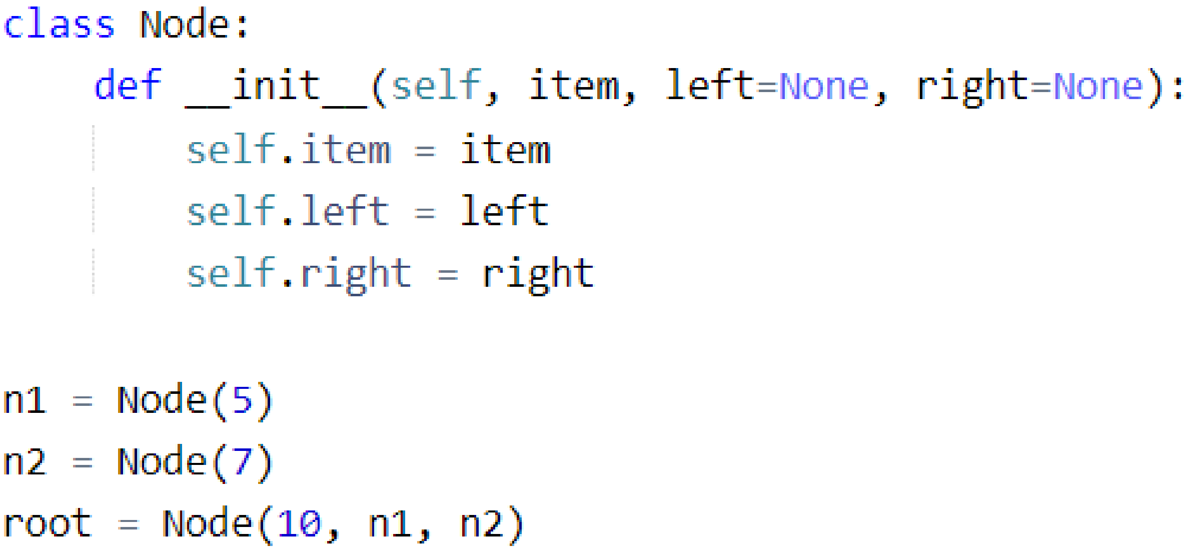
Height	Number of nodes at level h
$h = 0$	$2^0 = 1$
$h = 1$	$2^1 = 2$
$h = 2$	$2^2 = 4$

Total number of nodes in the tree:

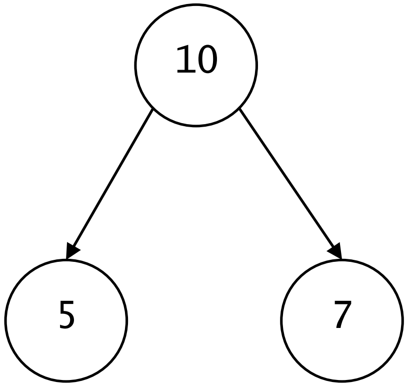
$$2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1$$

Tree: implementation


left item right


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

n1 = Node(5)
n2 = Node(7)
root = Node(10, n1, n2)
```



Tree: operations

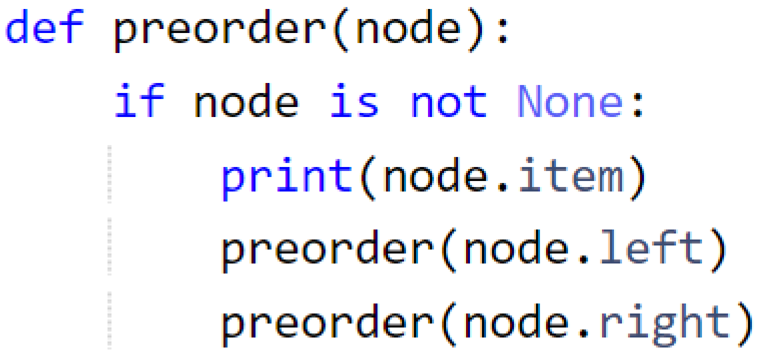
- **Basic operations**
    - Insertion
    - Deletion
    - Traversal
    - Search
- **Auxiliary operations**
    - Size of the tree
    - Height of the tree

Traversal

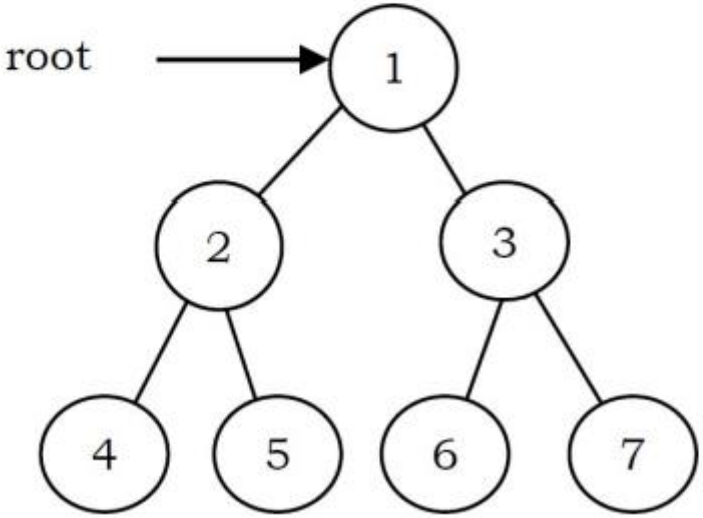
- **Depth First Traversal**
  - Preorder: self-left-right
  - Inorder: left-self-right
  - Postorder: left-right-self
- **Breadth First Traversal**
  - Level Order Traversal

Preorder Traversal

- **Self**-left-right
- Each node is processed before (pre) its subtrees



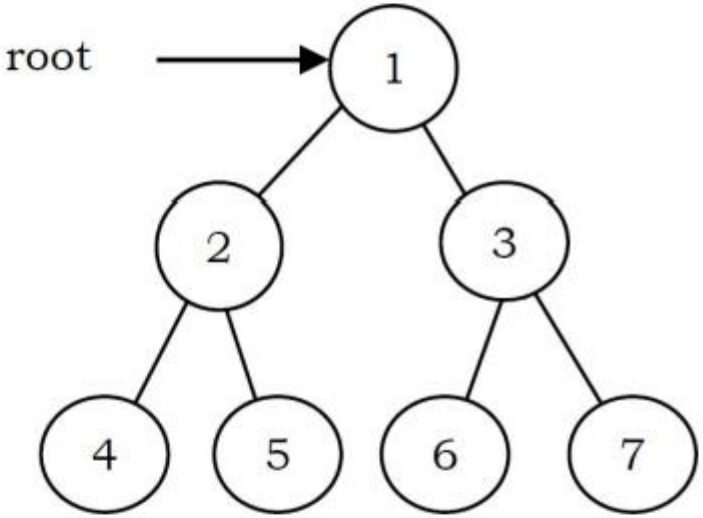
```python
def preorder(node):
    if node is not None:
        print(node.item)
        preorder(node.left)
        preorder(node.right)
```



Preorder: 1 2 4 5 3 6 7

Inorder Traversal

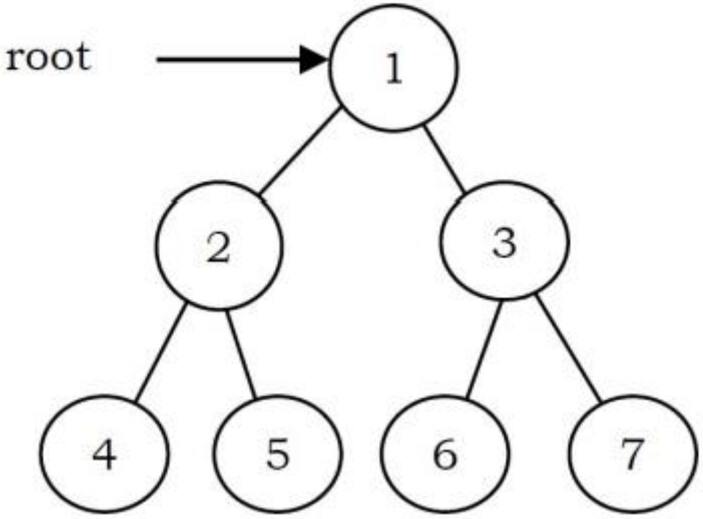
- Left-**self**-right
- Each node is processed (in) between its subtrees



Inorder: 4 2 5 1 6 3 7

Level Order Traversal

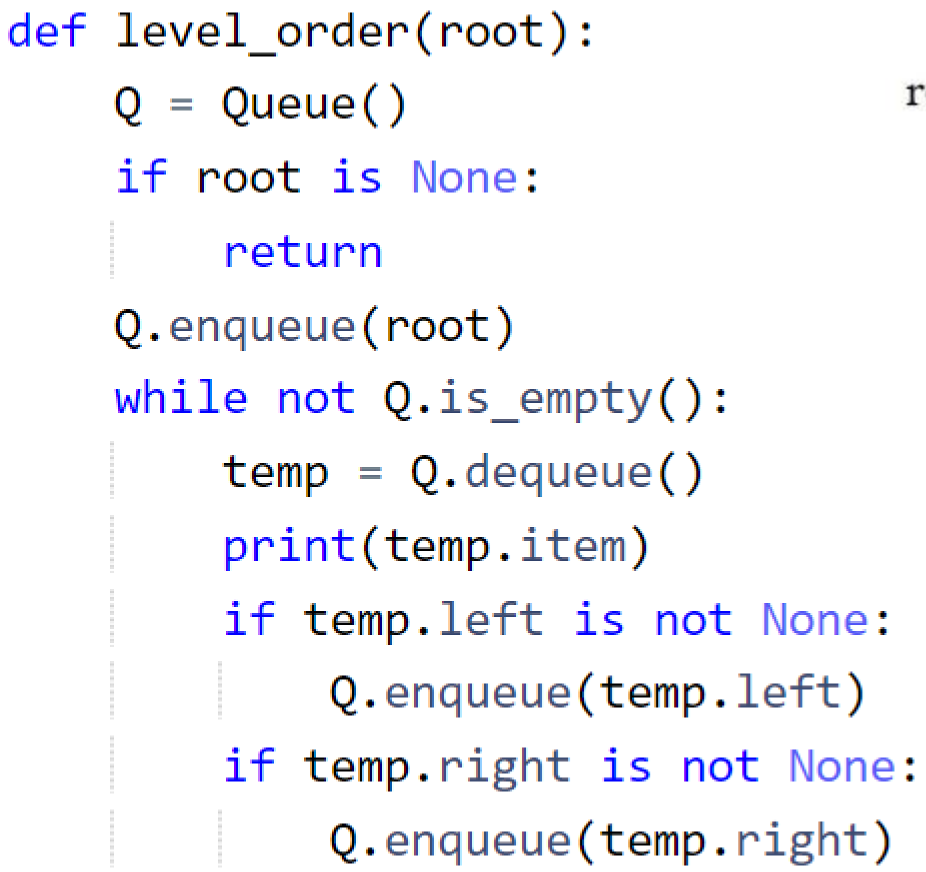
- Breadth First Traversal
  - While traversing a level h,
    - Keep track of nodes at the next level (h+1)
  - Go to the next level
    and visit all the nodes tracked nodes
  - Repeat this until
    all levels are completed



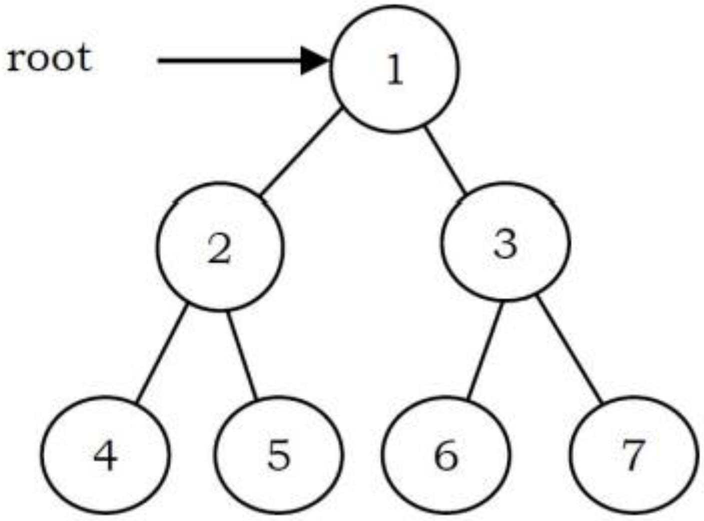
Level Order: 1 2 3 4 5 6 7

Level Order Traversal

- **Implementation with a Queue**



```
def level_order(root):
    Q = Queue()
    if root is None:
        return
    Q.enqueue(root)
    while not Q.is_empty():
        temp = Q.dequeue()
        print(temp.item)
        if temp.left is not None:
            Q.enqueue(temp.left)
        if temp.right is not None:
            Q.enqueue(temp.right)
```



Level Order: 1 2 3 4 5 6 7

Size, height, search

- Can you find size and height of a tree using traversal?
- Can you search a node holding item **x**?