Data Structure - Spring 2022
13. Tree (Lab)

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

TA: **Seong Joo Kim**

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

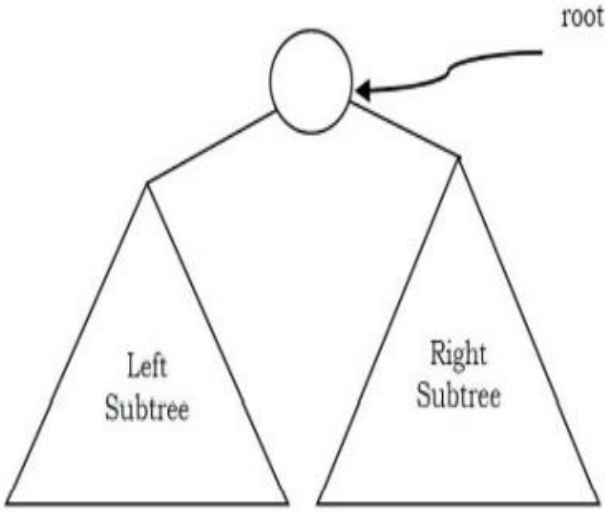
Computer Vision Lab
Hankuk University of Foreign Studies

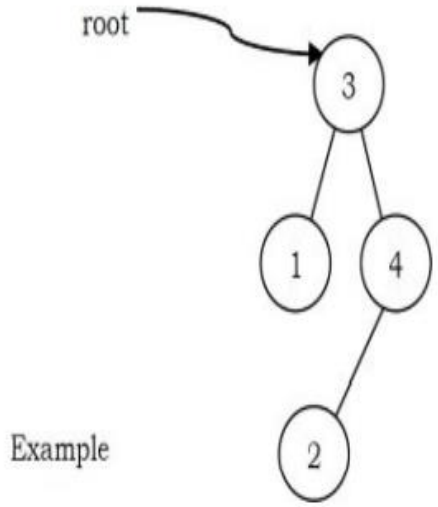
Today's Task

- Complete the basic Tree functions
  - **Traversal:** preorder, inorder, postorder
  - **Search**
  - **Insert_simple**: *already given in sample code*
  - **Size**
  - **Height**

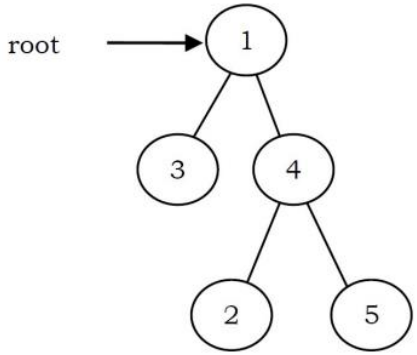
Binary Tree

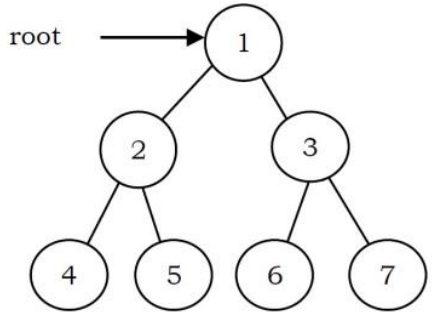
- Each node has $\leq$ 2 children

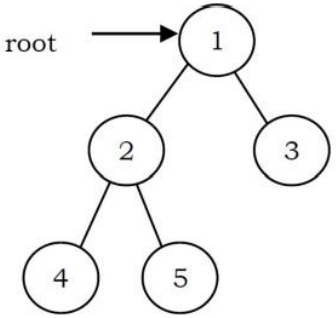




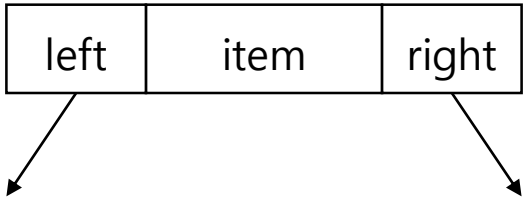
Example





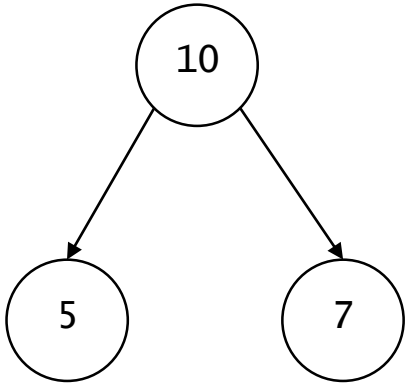


Tree: implementation




```
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)
```



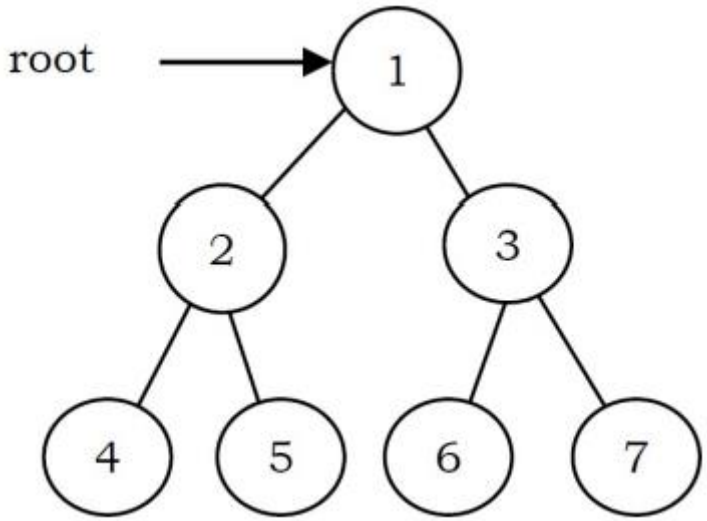
Traversal

- **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

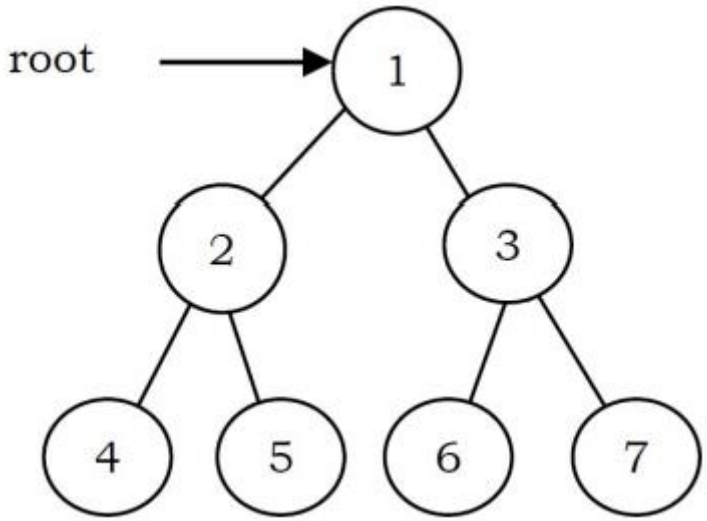
```
def preorder(node):
    if node is not None:
        print(node.item)
        preorder(node.left)
        preorder(node.right)
```



Preorder: 1 2 4 5 3 6 7

Inorder Traversal

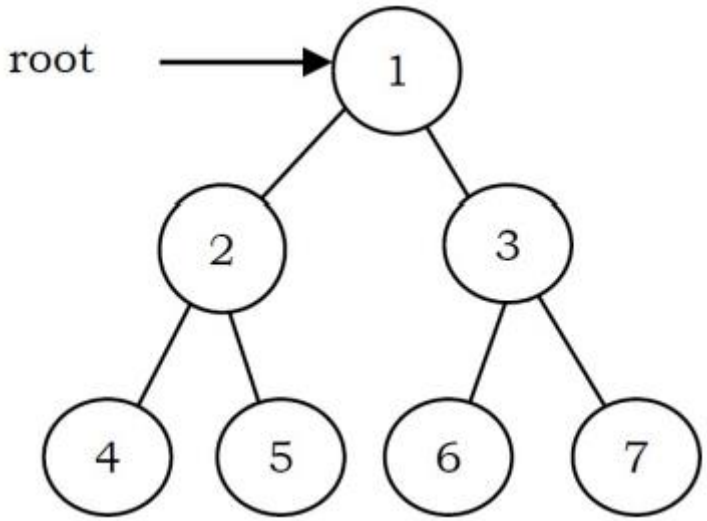
- Left-**self**-right
- Each node is processed (in) between its subtrees



Inorder: 4 2 5 1 6 3 7

Postorder Traversal

- Left-right-**self**
- Each node is processed after (post) its subtrees

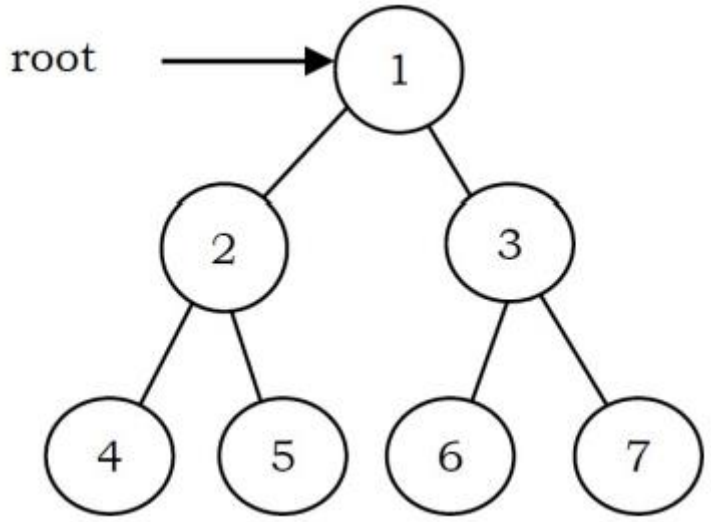


Inorder: 4 5 2 6 7 3 1

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

Search

- **Use preorder scheme**

- **search(root, x)**
  1. IF root is NONE: RETURN NONE
  2. IF root.item==x: RETURN root
  3. node = NONE
  4. IF root.left: node=search(root.left,x)
  5. IF node is not NONE: RETURN node
  6. IF root.right: node=search(root.right,x)
  7. RETURN node

Simple Insertion

- **insert_simple(root, p, side, x):**
  inserts **x** on the designated **side** of parent **p**

- This function is given in the sample code
- Main idea
  1. node_x = Node(x)
  2. node_p = search(root, p)
  3. IF side==left: node_p.left=node_x
  4. ELSE: node_p.right=node_x

Size and Height

- Use preorder traversal technique

- **size(root)**
  - IF root is NONE: RETURN 0
  - ELSE: RETURN 1+size(root.left)+size(root.right)

- **height(root)**
  - Formula: 1+max(height of left subtree, height of right subtree)
  - Note that:
  - In the right figure,
    - Height=2 (not 3)
    - If only one node (root only)
      - Height=0 (not 1)
    - If no node (empty tree/root is None)
      - Height can be considered to be (**-1**)