Data Structures in Python

- Tree Introduction
- Tree Traversals
- **Tree Algorithms**
- Binary Search Tree

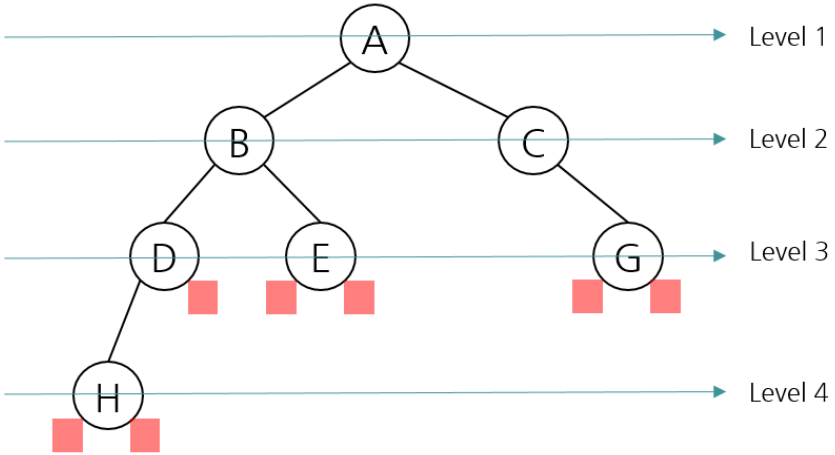
Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

Agenda & Readings

- Binary Tree Algorithms
  - levelorder() - Level order traversal
  - size()
  - height()
  - contains() - search
- Reference:
  - Problem Solving with Algorithms and Data Structures
  - Chapter 6 - Tree

levelorder()

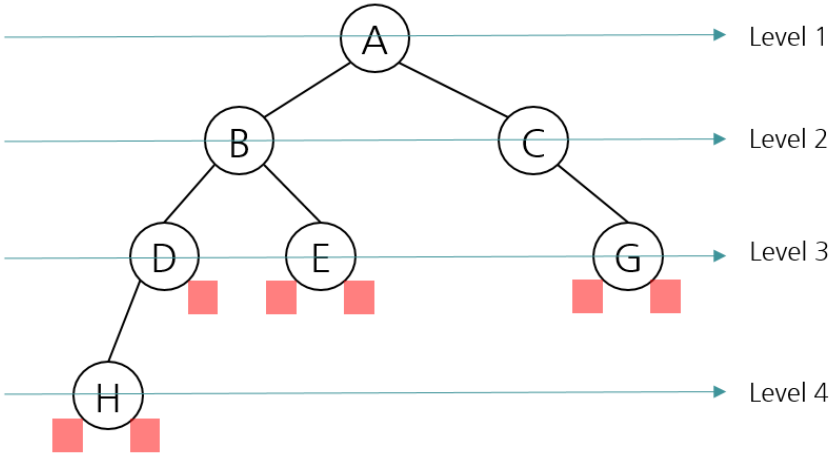
- The level order traversal traverses in level-order, where it visit every node one a level before going to a lower level.
- This search is referred as breadth-first search(BFS), as the search tree broadened as much as possible on each depth before going to the next depth.
- The time complexity can be either $O(n)$ and $O(n^2)$ depending on its implementation. Let us review the code with $O(n^2)$ time complexity. The coding of $O(n)$ algorithm using a queue-like structure is left as an exercise.



levelorder() - $O(n^2)$

- **Step 1:** Let us implement `printlevel(node, level)` that prints node's keys at a given level. It returns True if it prints a node's key, False otherwise.

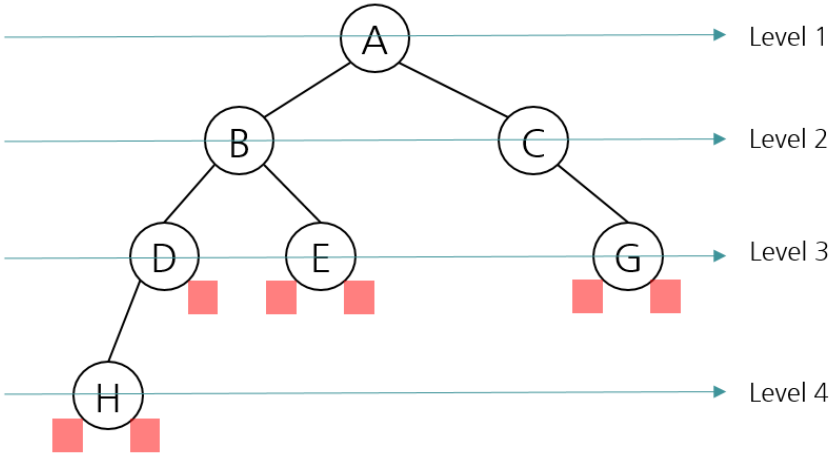
```
def printlevel(node, level):
    if node is None:                # base case
        return False
    if level == 1:
        print(node.key, end=' ')
        return True                 # return true if at least one node is present
    left  = printlevel(node.left, level - 1)
    right = printlevel(node.right, level - 1)
    return left or right
```



levelorder() - $O(n^2)$

- **Step 1:** Let us implement `printlevel(node, level)` that prints node's keys at a given level. It returns True if it prints a node's key, False otherwise.

```
def printlevel(node, level):
    if node is None:                # base case
        return False
    if level == 1:
        print(node.key, end=' ')
        return True                 # return true if at least one node is present
    left  = printlevel(node.left, level - 1)
    right = printlevel(node.right, level - 1)
    return left or right
```



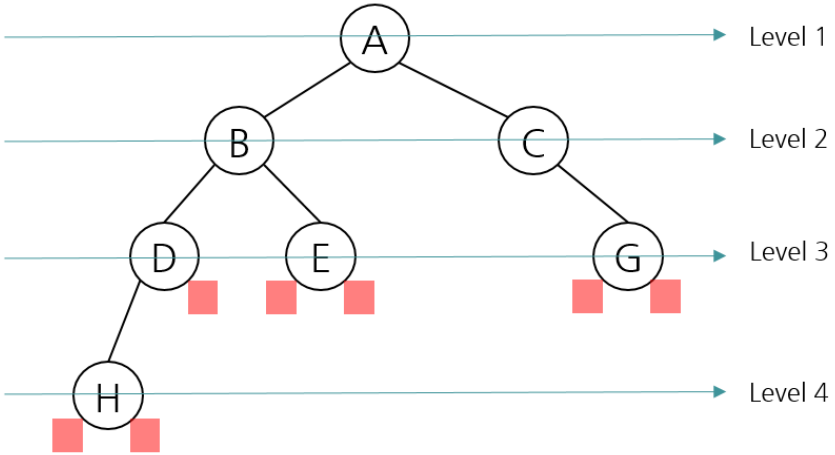

```
if __name__ == '__main__':
    print(printlevel(root, 1))
    print(printlevel(root, 2))
    print(printlevel(root, 3))
    print(printlevel(root, 5))
```

```
A True
B C True
D E G True
False
```

levelorder() - $O(n^2)$

- **Step 2:** We just keep on invoking `printlevel()` starting from `level 1` until the function returns False. The function return False when there is no node exists at a level.

```
def printlevel(node, level):
    if node is None:                # base case
        return False
    if level == 1:
        print(node.key, end=' ')
        return True                 # return true if at least one node is present
    left  = printlevel(node.left, level - 1)
    right = printlevel(node.right, level - 1)
    return left or right
```



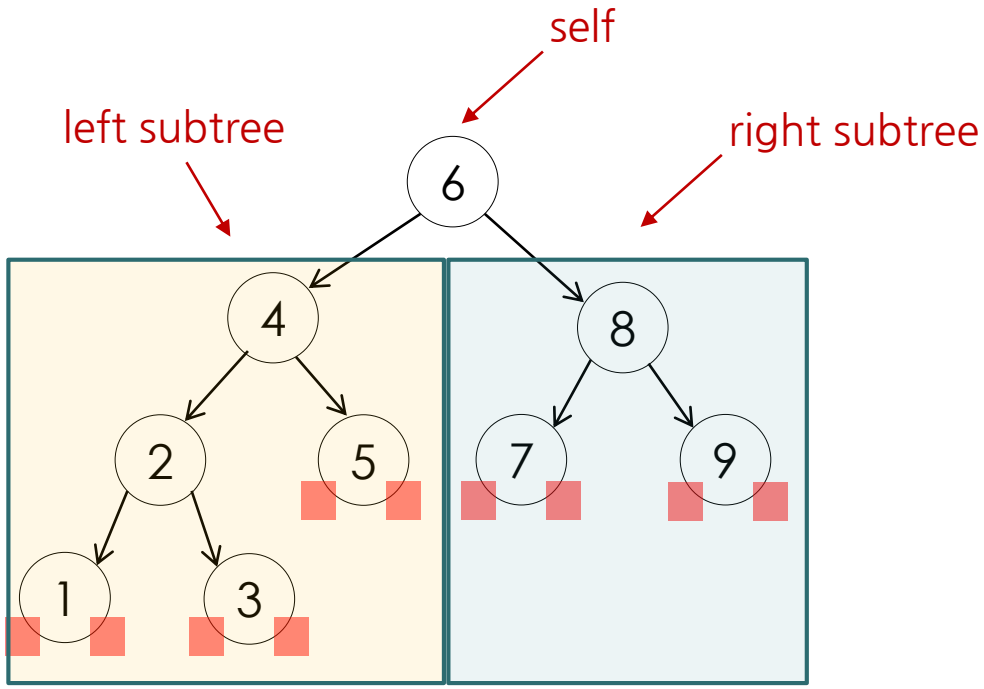

```
def levelorder_print(node):
    level = 1
    while printlevel(node, level):
        level = level + 1
        print()

if __name__ == '__main__':
    levelorder_print(root)
```

```
A
B C
D E G
H
```

size()

- tree size = size of left subtree + size of right subtree + 1.

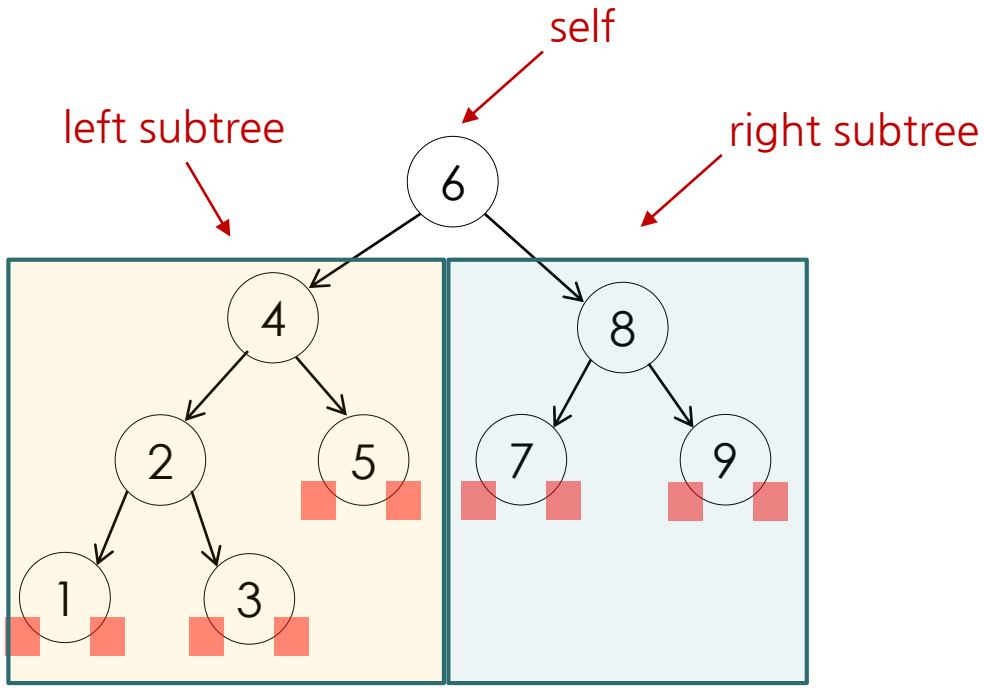
```
def size(node):
    if node is None:        # base case
        return 0
    return size(node.left) + size(node.right) + 1
```



size()

- tree size = size of left subtree + size of right subtree + 1.

```
def size(node):
    if node is None:        # base case
        return 0
    return size(node.left) + size(node.right) + 1
```



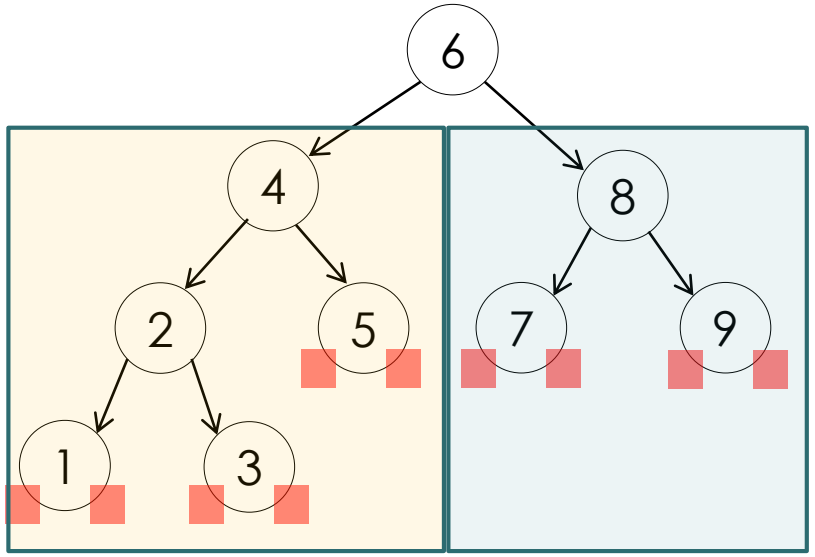
Questions:

1. What is the total number of the recursive function calls made to finish the initial call?
2. Which node invokes the last function call?
3. Which node finishes its size function call and returns size = 1 for the first time?

height()

- height = max(height of left subtree, height of right subtree) + 1.

```
def height(node):
    if node is None:       # base case
        return -1          # -1 if empty tree
    left  = height(node.left)
    right = height(node.right)
    return max(left, right) + 1
```



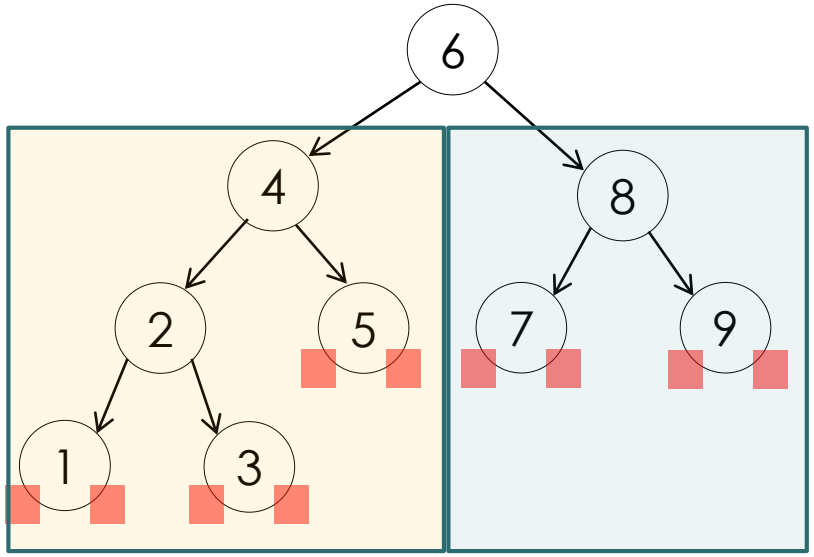
Questions:

1. What is the total number of the function call to complete with the tree?
2. What is the return value of the 10th function call?
3. What is the return value of the node 4?

contains()

- contains()

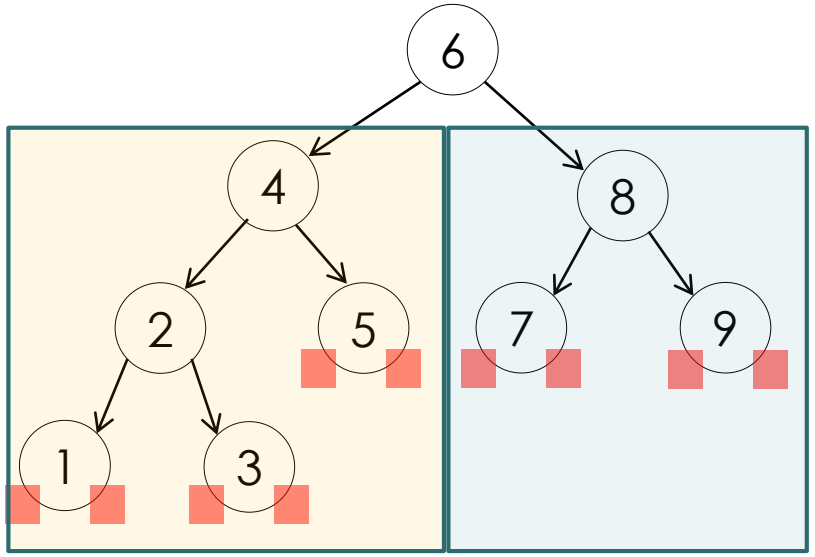
```
def contains(node, key):
    if node is None: return False
    if node.key == key: return True

    return contains(node.left, key) or contains(node.right, key)
```



contains()

- contains()

```
def contains(node, key):
    if node is None: return False
    if node.key == key: return True

    return contains(node.left, key) or contains(node.right, key)
```



Questions:

1. Which node invokes `contains(root.right, key)` for the first time?
2. Which node will invoke return False for the first time?
3. How many function calls are made to reach the node key=5?
4. How many function calls still remains in the system stack to finish after key=5 is found and what are they?

Summary

- The Level order traversal is a breadth-first search. The time complexity of the level order traversal algorithm may be either $O(n)$ or $O(n^2)$.
- size(): size = size of left subtree + size of right subtree + 1.
- height(): height = max(height of left subtree, height of right subtree) + 1.
- contains(): search/find function

Data Structures in Python

- Tree Introduction
- Tree Traversals
- **Tree Algorithms**
- Binary Search Tree

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University