Data Structures in Python

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

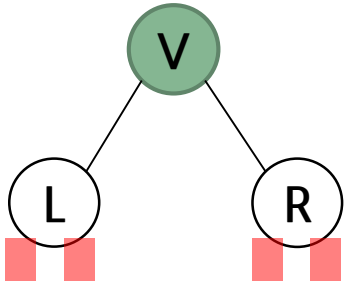
Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204, Handong Global University

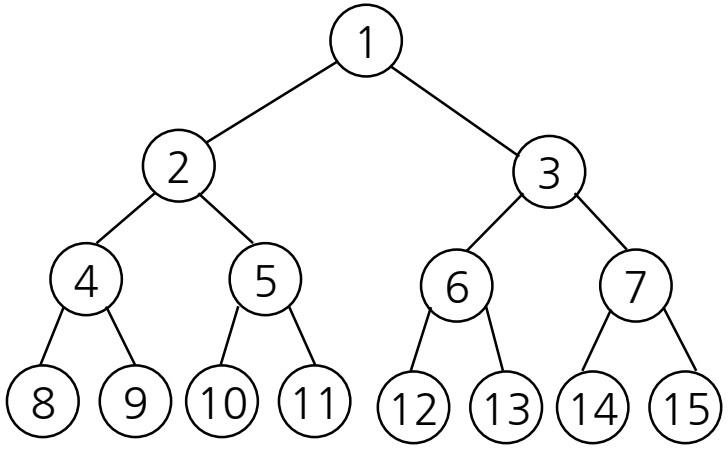
Agenda & Readings

- Binary Tree Traversals
  - Preorder
  - Inorder
  - Postorder
  - Level order
- Reference:
  - Problem Solving with Algorithms and Data Structures
  - Chapter 6 - Tree

Binary tree traversals

- **Tree traversal** (known as **tree search**) refers to the process of visiting each node in a tree, exactly once, in a systematic way.
- DFS (Depth-first Search)
  - There are three possible moves if we traverse left before right:
    LVR - inorder
    LRV - postorder
    VLR - preorder
  - They are named according to the position of **V**(the visiting node) with respect to the L and R.
  - These searches are referred to as **depth-first search(DFS)** since the search tree is deepened as much as possible on each child before going to the next sibling.



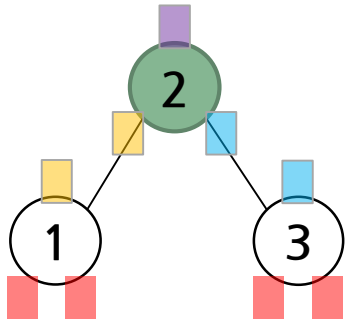


Inorder traversal(LVR) Example

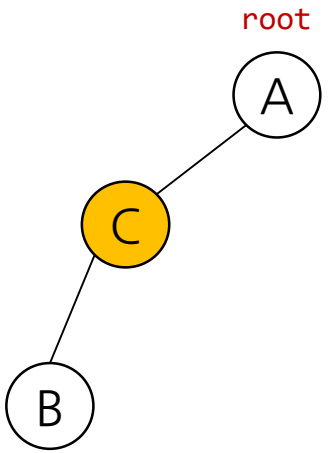
- Step 1 – Recursively traverse left subtree.
- Step 2 – Visit root node. (print or save it.)
- Step 3 – Recursively traverse right subtree.

```
def inorder(node):
    if node != None:
        inorder(node.left)          L
        print(node.key)             V
        inorder(node.right)         R
```

Is this a method in BinaryTree or an external function?



Output(LVR):
No. of inorder() calls made:



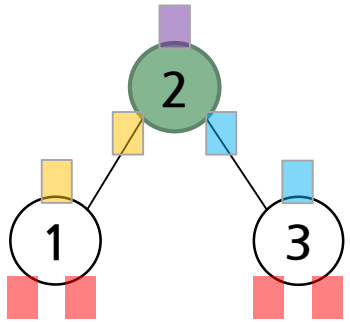
Output(LVR):
No. of inorder() calls made:

Inorder traversal(LVR) Example

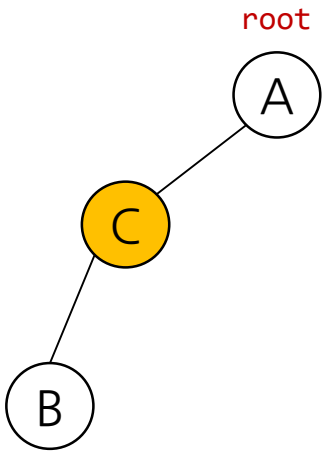
- Step 1 – Recursively traverse left subtree.
- Step 2 – Visit root node. (print or save it.)
- Step 3 – Recursively traverse right subtree.

```
def inorder(node):
    if node != None:
        inorder(node.left)          L
        print(node.key)             V
        inorder(node.right)         R
```

Is this a method in BinaryTree or an external function?



Output(LVR): 1 2 3
No. of inorder() calls made: 7

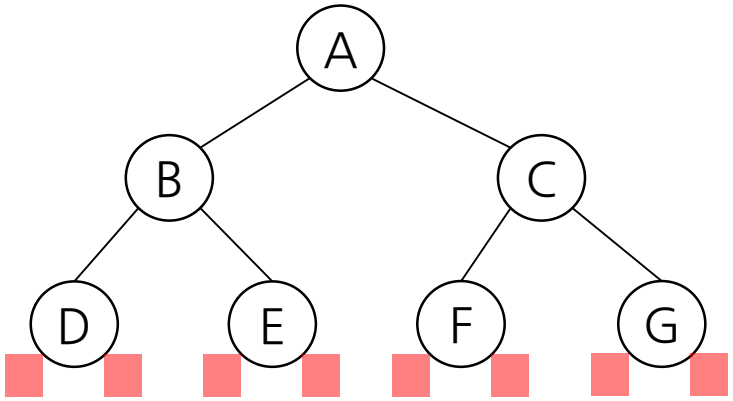


Output(LVR): B C A
No. of inorder() calls made: 7

Inorder traversal(LVR) Exercise

- Step 1 – Recursively traverse left subtree.
- Step 2 – Visit root node. (print or save it.)
- Step 3 – Recursively traverse right subtree.

```
def inorder(node):
    if node != None:
        inorder(node.left)          L
        print(node.key)             V
        inorder(node.right)         R
```

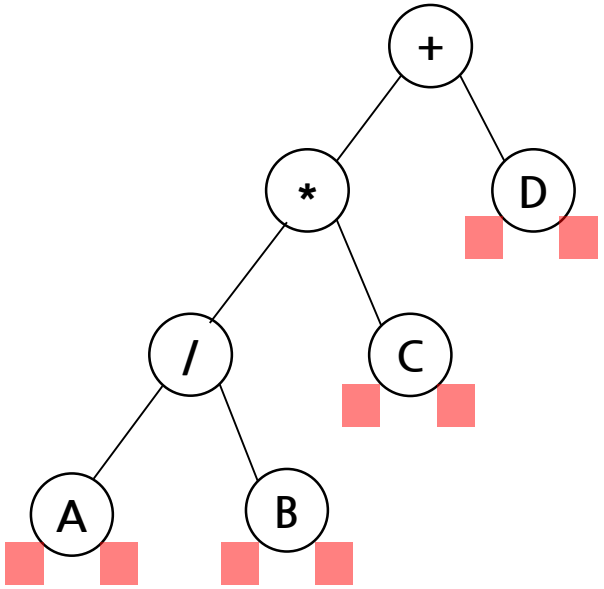


Output(LVR):
No. of inorder() calls made:

Inorder traversal(LVR) Exercise

- Step 1 – Recursively traverse left subtree.
- Step 2 – Visit root node. (print or save it.)
- Step 3 – Recursively traverse right subtree.

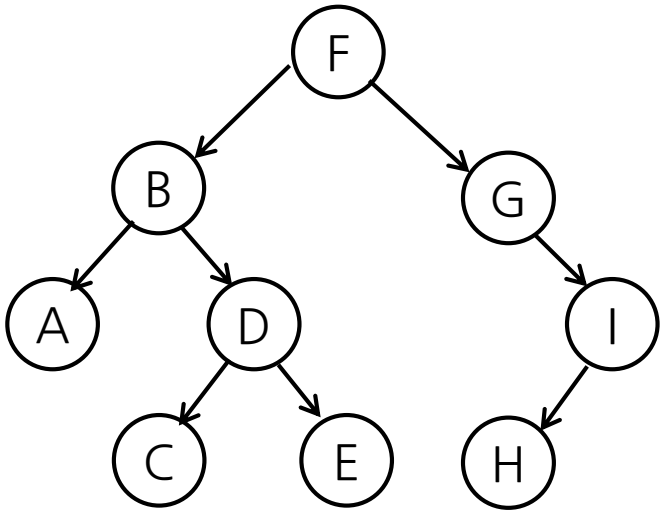
```
def inorder(node):
    if node != None:
        inorder(node.left)          L
        print(node.key)             V
        inorder(node.right)         R
```

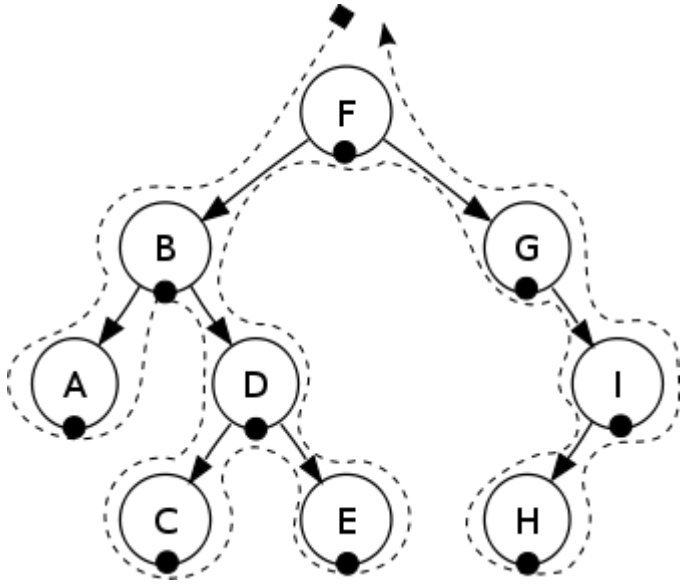


Output(LVR):
No. of inorder() calls made:

Inorder traversal(LVR) Exercise

- Traverse the left subtree.
- Visit the root.
- Traverse the right subtree.

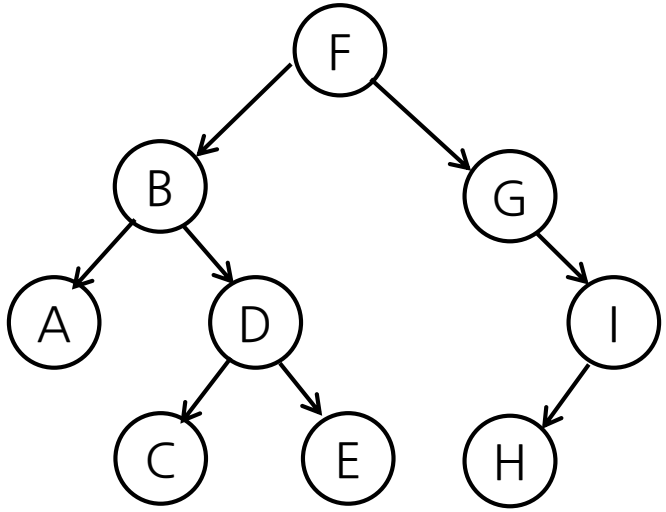




Output:

Preorder traversal(VLR) Example

- Step 1 – Visit root node.
- Step 2 – Recursively traverse left subtree.
- Step 3 – Recursively traverse right subtree.

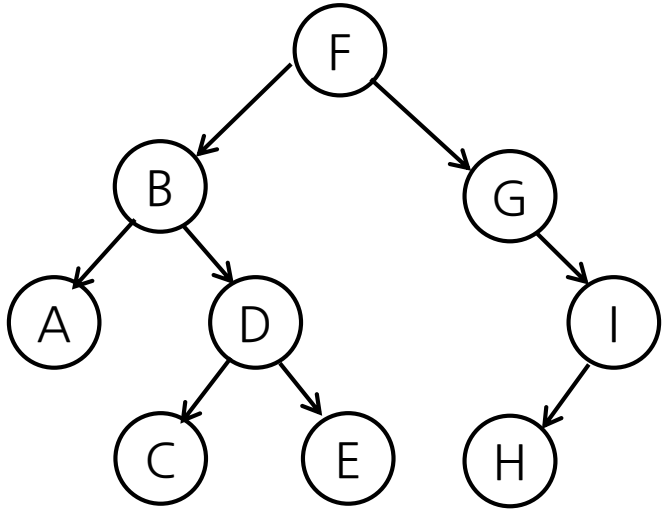


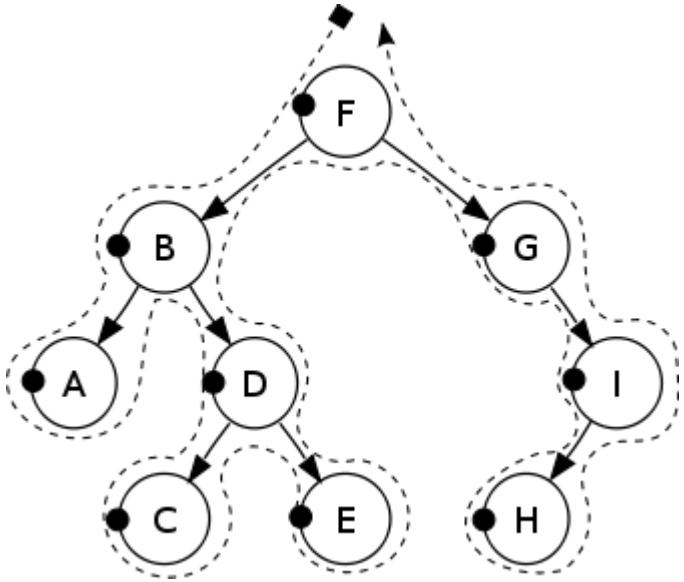

```
def preorder(node):
    if node != None:
        print(node.key)          V
        preorder(node.left)      L
        preorder(node.right)     R
```

Output(VLR):

Preorder traversal(VLR) Example

- Step 1 – **Visit root node**.
- Step 2 – Recursively traverse left subtree.
- Step 3 – Recursively traverse right subtree.



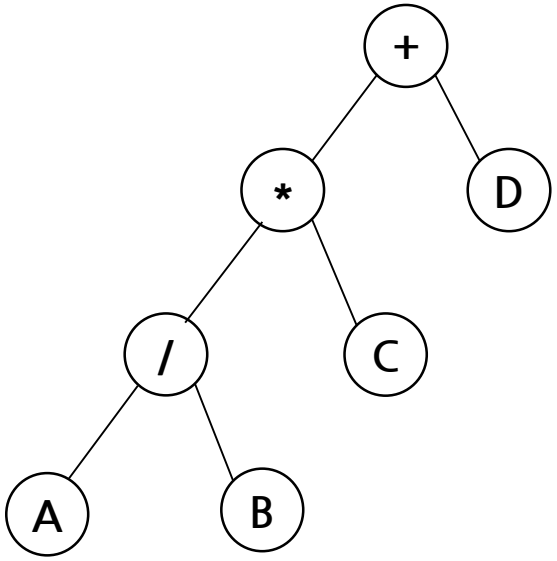



```
def preorder(node):
    if node != None:
        print(node.key)          V
        preorder(node.left)      L
        preorder(node.right)     R
```

Output(VLR): F, B, A, D, C, E, G, I, H

Preorder traversal(VLR) Exercise

- Step 1 – Recursively traverse left subtree.
- Step 2 – Recursively traverse right subtree.
- Step 3 – Visit root node.

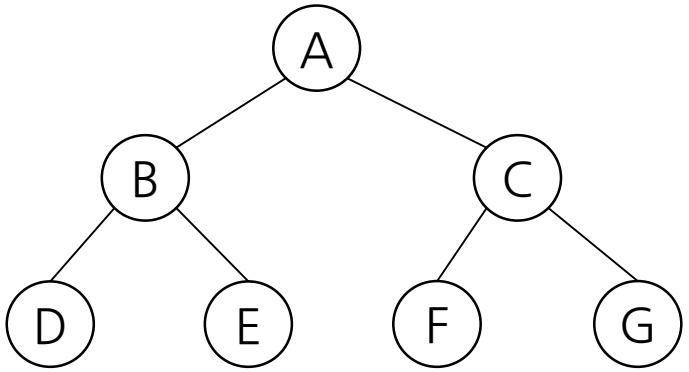



```
def postorder(node):
    if node != None:
        postorder(node.left)
        postorder(node.right)
        print(node.key)
```

Output(LRV):

Preorder traversal(VLR) Exercise

- Step 1 – Visit root node.
- Step 2 – Recursively traverse left subtree.
- Step 3 – Recursively traverse right subtree.

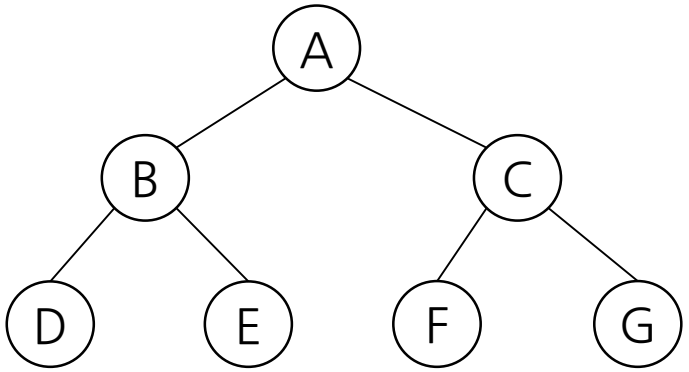



```
def preorder(node):
    if node != None:
        print(node.key)          V
        preorder(node.left)      L
        preorder(node.right)     R
```

Output(VLR):

Postorder traversal(LRV) Example

- Step 1 – Recursively traverse left subtree.
- Step 2 – Recursively traverse right subtree.
- Step 3 – Visit root node.

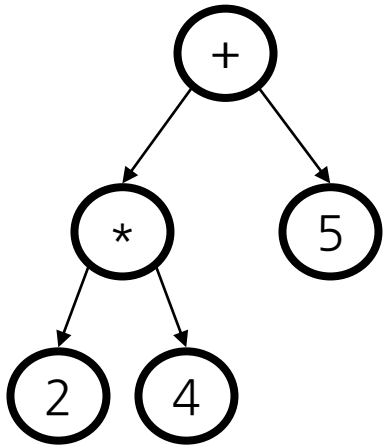



```
def postorder(node):
    if node != None:
        postorder(node.left)
        postorder(node.right)
        print(node.key)
```

Output(LR**V**): D E B F G C A

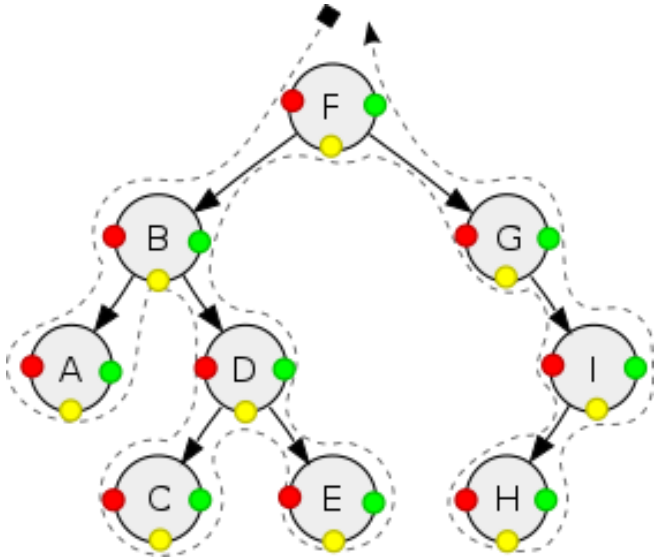
Binary tree traversals Exercise 1:

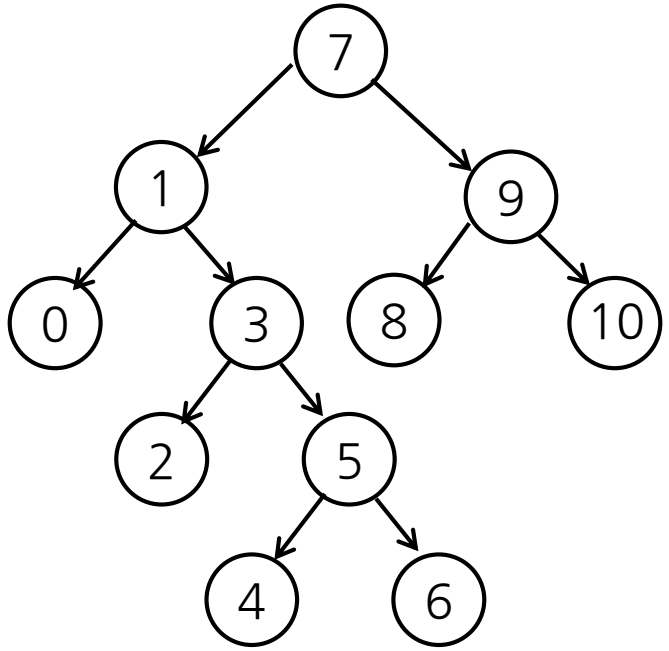
- preorder Traversal(VLR):
- inorder traversal(LVR) :
- postorder traversal(LRV):

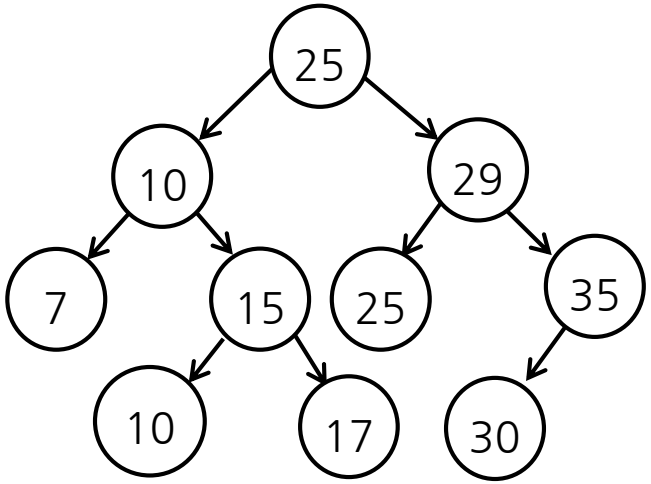


Binary tree traversals Exercise 2:

- preorder Traversal(VLR)
- inorder traversal(LVR)
- postorder traversal(LRV)







Binary tree traversals

- **Observations:**
  1. If you know you need to **explore the roots** before inspecting any leaves, you pick **preorder** because you will encounter all the roots before all of the leaves.
  2. If you know you need to **explore all the leaves** before any nodes, you select **postorder** because you don't waste any time inspecting roots in search for leaves.
  3. If you know that the tree has an inherent sequence in the nodes, and you want to flatten the tree back into its original sequence, then an **inorder** traversal should be used. The tree would be flattened in the same way it was created. A pre-order or post-order traversal might not unwind the tree back into the sequence which was used to create it.
  4. In a binary search tree ordered such that in each node the key is greater than all keys in its left subtree and less than all keys in its right subtree, **inorder traversal** retrieves the keys in ***ascending*** **sorted order**.

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