Traversals of a Binary Tree

Dr. Suleiman Abu Kharmeh

Tree Traversal

- Traversing a tree means visiting each node in a specified order
- This process is not as commonly used as finding, inserting, and deleting nodes. One reason for this is that traversal is not particularly fast.
- But traversing a tree is useful in some circumstances and the algorithm is interesting

- There are generally two types of traversal:
  - *breadth first*
  - *depth first.*
- There are three variants for depth first traverse a tree. They're called *preorder, inorder, and postorder.*

1-Inorder Traversal

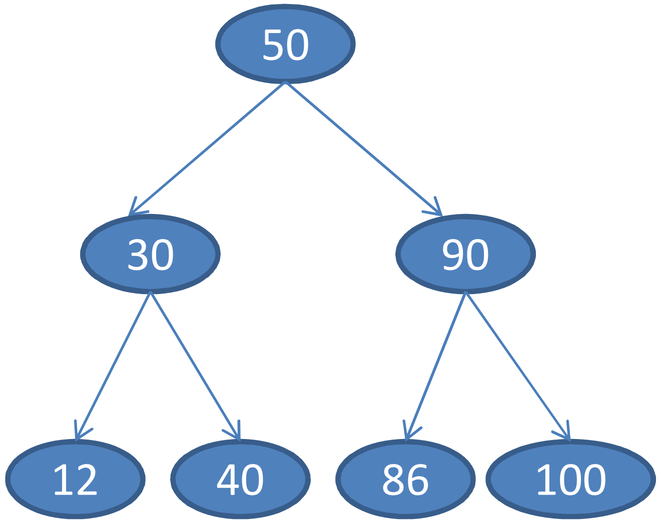
- An in-order traversal of a binary search tree will cause all the nodes to be visited *in ascending order, based on their key values. If you want to create a sorted list of the data in a binary tree,* this is one way to do it.

- The simplest way to carry out a traversal is the use of recursion . A recursive method to traverse the entire tree is called with a node as an argument. Initially, this node is the root. The method needs to do only three things:

1. Call itself to traverse the node's left subtree
2. Visit the node
3. Call itself to traverse the node's right subtree

Java Code for Traversing

```
private void inOrder(node localRoot)
{
    if(localRoot != null)
    {
        inOrder(localRoot.leftChild);
        localRoot.displayNode();
        inOrder(localRoot.rightChild);
    }
}
```



Determine the order in which the elements would be accessed during an in-order traversal.

- Answer: 12, 30, 40, 50, 86, 90, 100

2-Postorder

1. Call itself to traverse the node's left subtree.
2. Call itself to traverse the node's right subtree.
3. Visit the node.

Left – Right – Root

```
postOrder (t)
{
    if (t is not empty)
    {
        postOrder(leftTree(t));
        postOrder(rightTree(t));
        access the root element of t;
    } // if
} // postOrder traversal
```

3-Preorder

1. Visit the node.
2. Call itself to traverse the node's left subtree.
3. Call itself to traverse the node's right subtree.

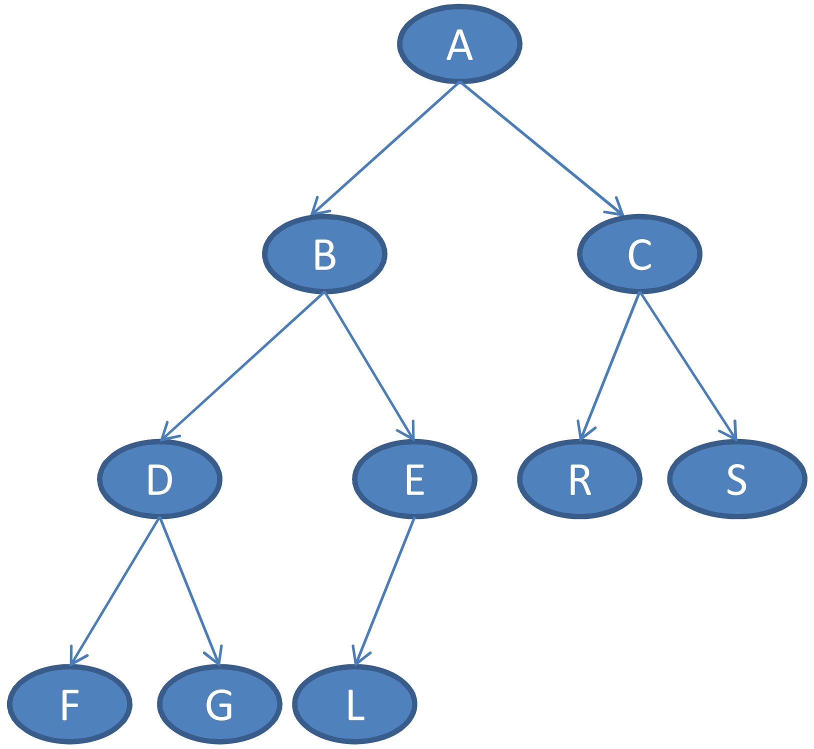
Root – Left – Right

```
preOrder (t)
{
    if (t is not empty)
    {
        access the root element of t;
        preOrder (leftTree (t));
        preOrder (rightTree (t));
    } // if
} // preOrder traversal
```

Breadth-first traversal (level order)

To perform a breadth-first traversal of a non empty binary tree, first access the root element, then the children of the root element, from left to right, then the grandchildren of the root element, from left to right, and so on.

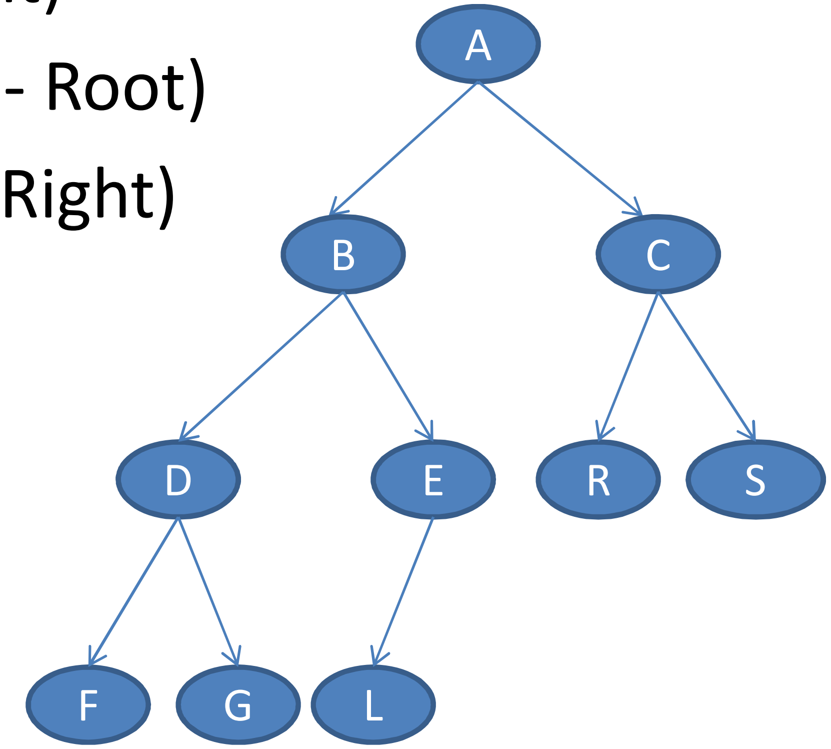
Perform a breadth-first traversal of the following binary tree.



- Answer: A, B, C, D, E, R, S, F, G, L

Exercise 1

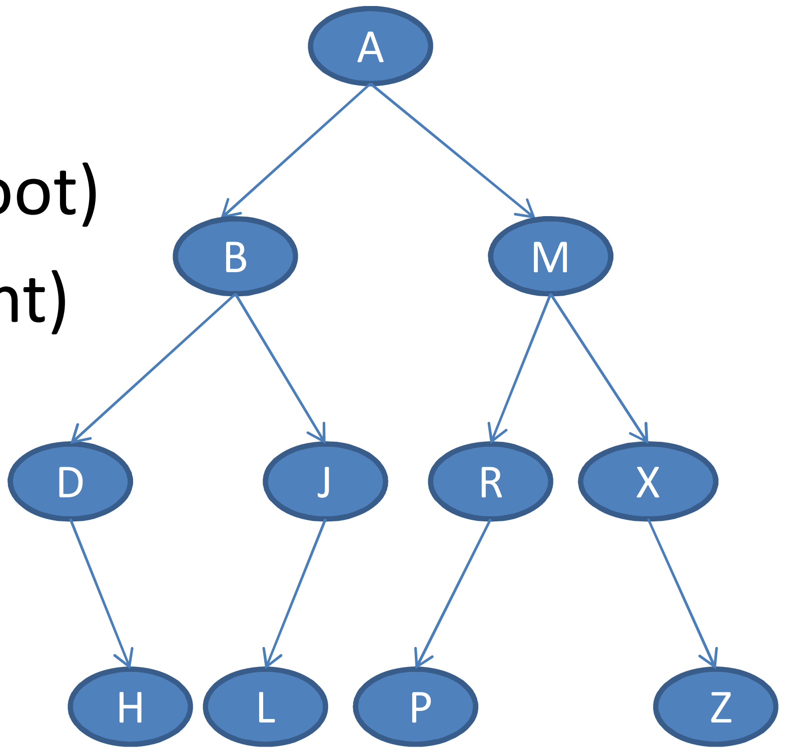
- Perform the other traversals of that tree:
  - inOrder (Left-Root-Right)
  - postOrder (Left - Right - Root)
  - preOrder (Root - Left - Right)



- In: F, D, G, B, L, E, A, R, C, S
- Post: F, G, D, L, E, B, R, S, C, A
- Pre: A, B, D, F, G, E, L, C, R, S

Exercise 2

- Perform the other traversals of the following tree:
  - inOrder (Left-Root-Right)
  - postOrder (Left - Right - Root)
  - preOrder (Root - Left - Right)


A
B
M
D
J
R
X
H
L
P
Z