Lecture Notes – Binary Tree Traversals

Binary tree traversals refer to the predetermined sequence we visit each node of the tree to be processed once and only once. The 2 general approaches to the traversal sequence are depth first and breadth first traversals.

Depth First Traversal

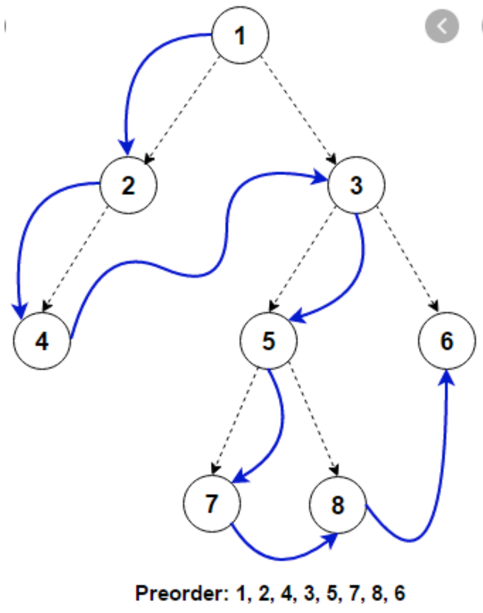
The processing proceeds along a path from the root through one child to the most distance descendent of that first child before processing a second child. i.e. you process all the descendants of a child before going on to the next child.
We will examine 3 different depth first traversals:

1. preorder traversal
2. inorder traversal
3. postorder traversal

Breadth First Traversal

The processing proceeds horizontally from the root to all of its children, then to its children's children, and so forth until all nodes have been processed. In other words, in the breadth first traversal, each level is completely processed before the next level is started.

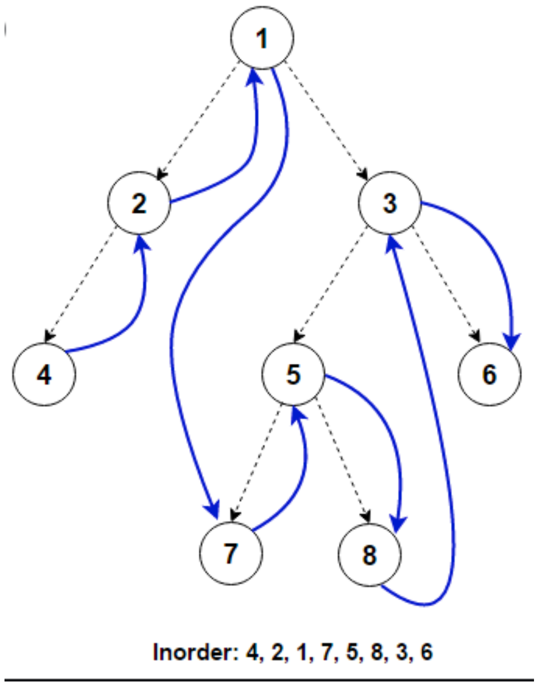
Preorder Traversal - (Rt-L-R)



```
def preorder(tree):
     if tree != None:
       print(tree.getRootVal())
       preorder(tree.getLeftChild())
       preorder(tree.getRightChild())
```

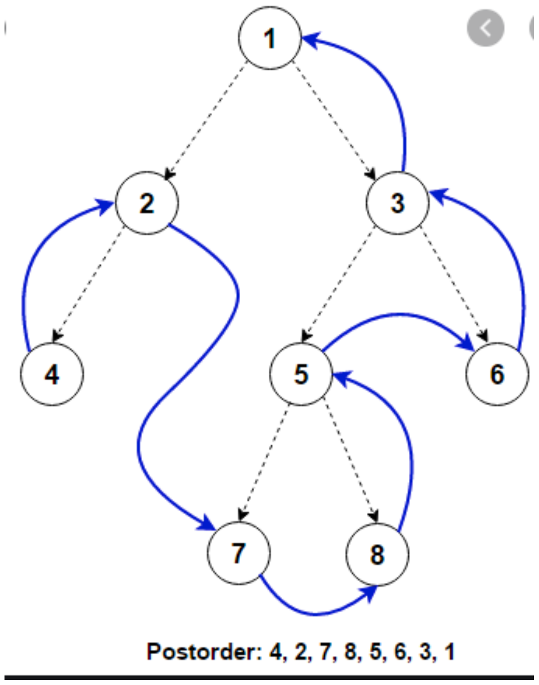
Preorder, inorder, postorder traversal images from techiedelight.com

Inorder Traversal – (L-RT-R)



```
def inorder(tree):
   if tree != None:
       inorder(tree.getLeftChild())
       print(tree.getRootVal())
       inorder(tree.getRightChild())
```

Postorder Traversal – (L-R-RT)



```
def postorder(tree):
     if tree != None:
       postorder(tree.getLeftChild())
       postorder(tree.getRightChild())
       print(tree.getRootVal())
```

Preorder, inorder, postorder traversal images from techiedelight.com

For practice:
Consider the following tree. What are the preorder, inorder, postorder traversals?

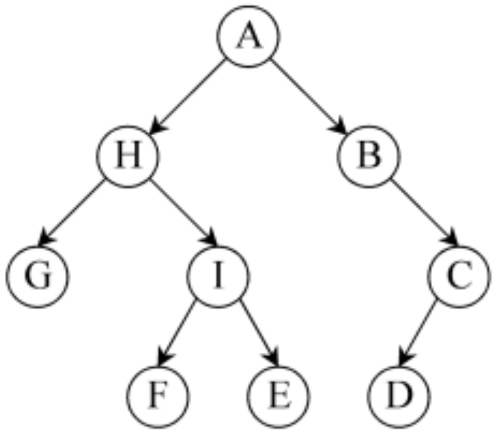


image from commons.wikimedia.org

preorder:	A H G I F E B C D
inorder:	G H F I E A B D C
postorder:	G F E I H D C B A

Breadth First Traversal

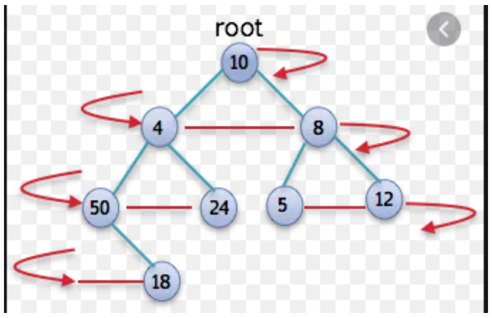


Image from Kodebinary.com

```
Q ->  10
Q ->   4  8
Q ->   8  50  24
Q ->  50 24 5 12
Q ->  24 5 12 18
Q ->  5 12 18
Q ->  12 18
Q ->  18
Q ->  ^
```

```
ptr = root
loop (ptr != None)
        process(ptr)
        if (ptr.getLeftChild != None)
                Q.enqueue (ptr.getLeftChild())
        if (ptr.getRightChild != None)
                Q.enqueue (ptr.getRightChild())
        if  not Q.isempty()
                dequeue (ptr)
        else
                ptr = None
endloop
```