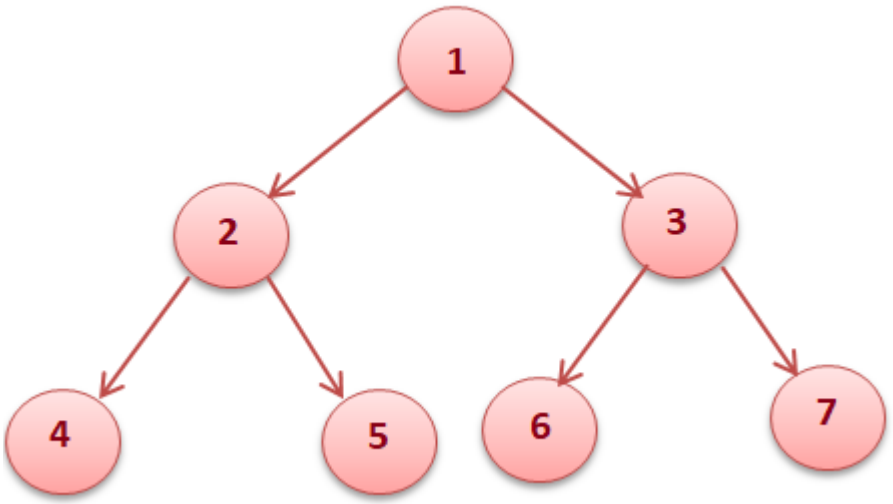
Binary Tree Traversals

The process of visiting all the nodes of the tree is called tree traversal. Each node is processed only once but may be visited more than once. We will be considering the below tree for all the traversals.

D: Current node

L: Left Subtree

R: Right Subtree



- **PreOrder Traversal (DLR)**

In preorder traversal, each node is processed before processing its subtrees.

Like in the above example, 1 is processed first, then the left subtree, and this is followed by the right subtree. Therefore processing must return to the right subtree after finishing the processing of the left subtree. To move to the right subtree after processing the left subtree we have to maintain the root information.

Steps for preOrder traversal

- Visit the root.

- Traverse the left subtree in Preorder.
- Traverse the right subtree in Preorder.

```
function preOrderTraversal(root)

        if root is null
              return
  //   process current node
        print “root.data”
        //   calling function recursively for left and right subtrees
        preOrderTraversal(root.left)
        preOrderTraversal(root.right)
```

PreOrder Output of above tree : 1 2 4 5 3 6 7

- **InOrder Traversal (LDR)**

In inorder traversal, the root is visited between the subtrees.

Steps for InOrderTraversal

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

```
function inOrderTraversal(root)
        if root is null
              return

        //   calling function recursively for left subtree
        inOrderTraversal(root.left)
        //   process current node
        print “root.data”
        //   calling function recursively for right subtree
        inOrderTraversal(root.right)
```

InOrder Output of above tree : 4 2 5 1 6 3 7

- **PostOrder Traversal(LRD)**

In postorder traversal, the root is visited after the left and right subtrees respectively.

Steps for postOrder traversal

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

```
function postOrderTraversal(root)
        if root is null
              return

        //   calling function recursively for left and right subtrees
        postOrderTraversal(root.left)
        postOrderTraversal(root.right)
        //   process current node
        print “root.data”
```

postOrder output of above tree : 4 5 2 6 7 3 1

- **LevelOrder Traversal**

In level order traversal, each node is visited level-wise in increasing order of levels from left to right.

Steps for levelOrder traversal

- Visit the root.
- While traversing the level l, add all the elements at (l + 1) in the queue
- Go to the next level and visit all the nodes at that level.
- Repeat this until all levels are completed.

```
function levelOrderTraversal()
        if root is null
              return

        //   Create queue of type Node and add root to the queue
        queue.add(root).

        while the queue is not empty
              //   remove the front item from the queue
              removedNode= queue.remove()
         //   process the removedNode
            print “removeNode.data”
```

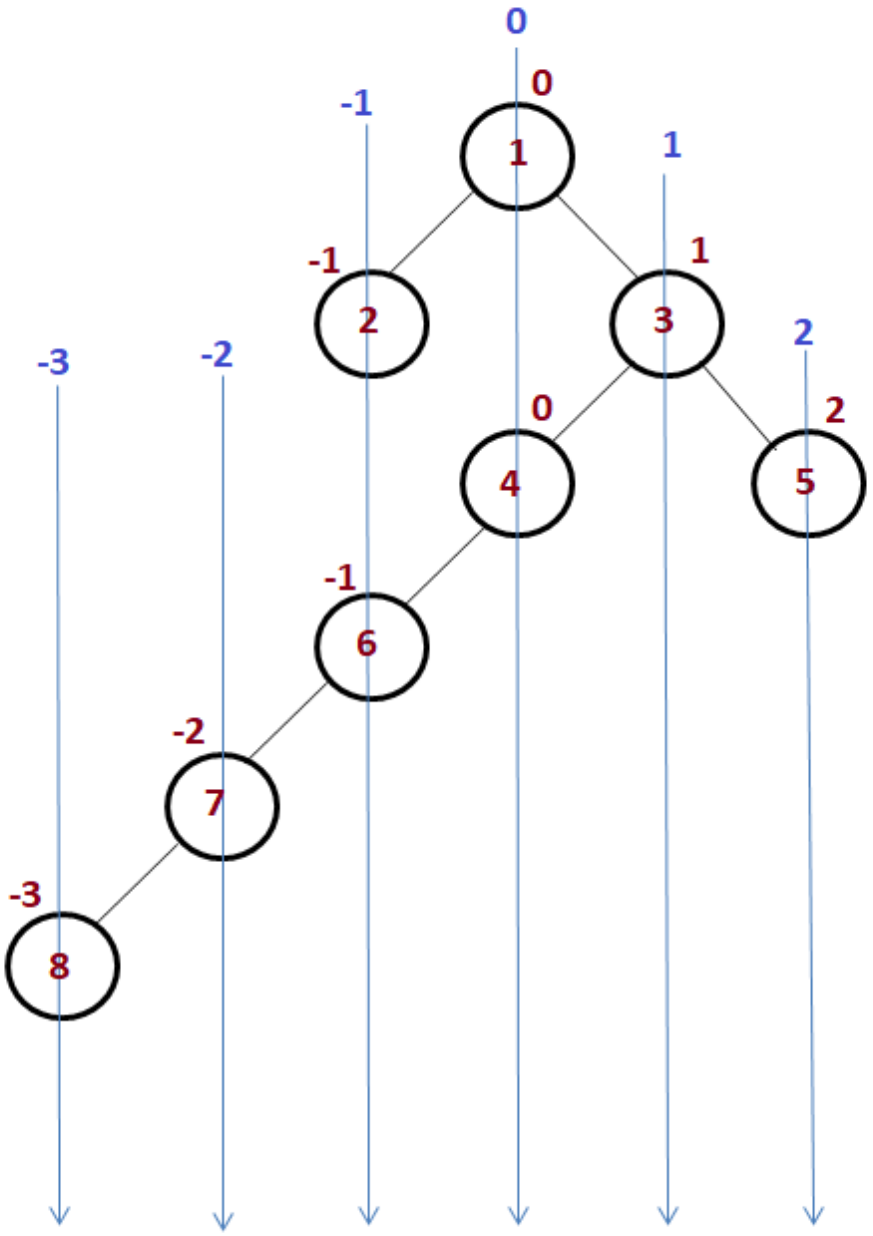
```
            /*
          Add the left and right child of the removedNode if they are not null
            */

            if removedNode.left is not null
                    queue.add(removedNode.left)
            if removedNode.right is not null
              queue.add(removedNode.right)
end
```

LevelOrder output of above tree: 1 2 3 4 5 6 7

Follow Up: What if we want to print each level in a new line.

Maintain 2 queues, primaryQueue, and secondaryQueue, and add new elements for the next level in the secondaryQueue. When your primaryQueue becomes empty, make primaryQueue as secondaryQueue, print a new line, and reinitialize the secondaryQueue as a new Queue.

- **Vertical Order Traversal**



Output: 8 7 2 6 1 4 3 5

Note: If two nodes are at the same horizontal distance like 2 and 6 then 2 must be printed before 6 i.e mentioning the order

Steps for vertical Order traversal

- Create a TreeMap of the vertical level Vs list integers(in that particular vertical level)
- Maintain another variable with each node know as the vertical level of that node
- Visit the root. Add it to the queue.
- Remove the node from the queue say removedNode and add the node's value in treemap across the removedNode.verticallevel.
- While traversing level l, add left and right child of the removed node to queue and treemap setting the leftNode's and rightNode's vertical level equal to removedNode.verticalLevel-1 and removedNode.verticalLevel+1 respectively.
- Go to the next level and visit all the nodes at that level.
- Repeat this until all levels are completed.
- After this traversal, iterate over the treemap, get the list across the current key of the treemap and print the list.

```
function verticalOrderTraversal()
        if root is null
         return

        //   Create queue of type Node and add root to the queue
        queue.add(root).

        while the queue is not empty
           //   remove the front item from the queue
              removedNode= queue.remove()
         //   process the removedNode, add to the list in treemap across that level
              levelItems = map.get(root.verticalLevel)
         levelItems.add(removedNode.data)

              /*
          Add the left and right child of the removedNode if they are not null
              */

              if removedNode.left is not null
                removedNode.left.verticalLevel = removedNode.verticalLevel-1
            queue.add(removedNode.left)
              if removedNode.right is not null
                removedNode.right.verticalLevel = removedNode.verticalLevel+1
            queue.add(removedNode.right)
  end

  //   Iterate over the treemap
  for(every Key in map)
        print(map.get(key))
```