THE BINARY TREE - TRAVERSAL

WHAT'S INTERESTING ABOUT
BINARY TREES IS THE THAT THERE
ARE A WHOLE NUMBER OF WAYS
TO VISIT THE NODES OF A TREE

THEY VARY BASED ON THE ORDER
IN WHICH THE NODES ARE
ACCESSED

VISITING NODES OF A TREE IS
CALLED TRAVERSING A TREE

BREADTH-FIRST

DEPTH-FIRST

BREADTH-FIRST TRAVERSAL

BREADTH FIRST TRAVERSAL INVOLVES VISITING NODES AT EVERY LEVEL BEFORE MOVING ON TO THE NEXT LEVEL

START WITH THE ROOT NODE - IT'S AT LEVEL 0 AND IS THE FIRST NODE TO VISIT

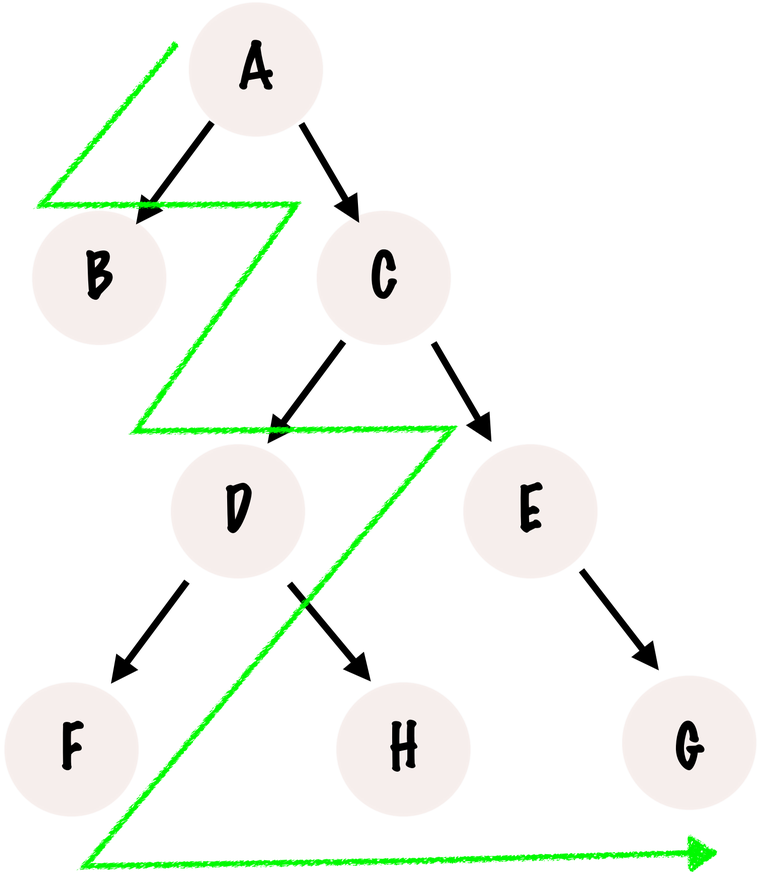
THE NEXT STEP IS TO CHECK WHETHER THERE ARE OTHER NODES AT THE SAME LEVEL AND VISIT THEM

ONCE A LEVEL IS EXHAUSTED THEN WE MOVE TO THE NEXT LEVEL

WE CONTINUE THIS TILL EVERY NODE OF THE TREE HAS BEEN VISITED

BREADTH-FIRST TRAVERSAL

THE PATH THAT BREADTH-FIRST TRAVERSAL TAKES



LEVEL N

↓

LEVEL N+1

↓

LEVEL N+2

↓

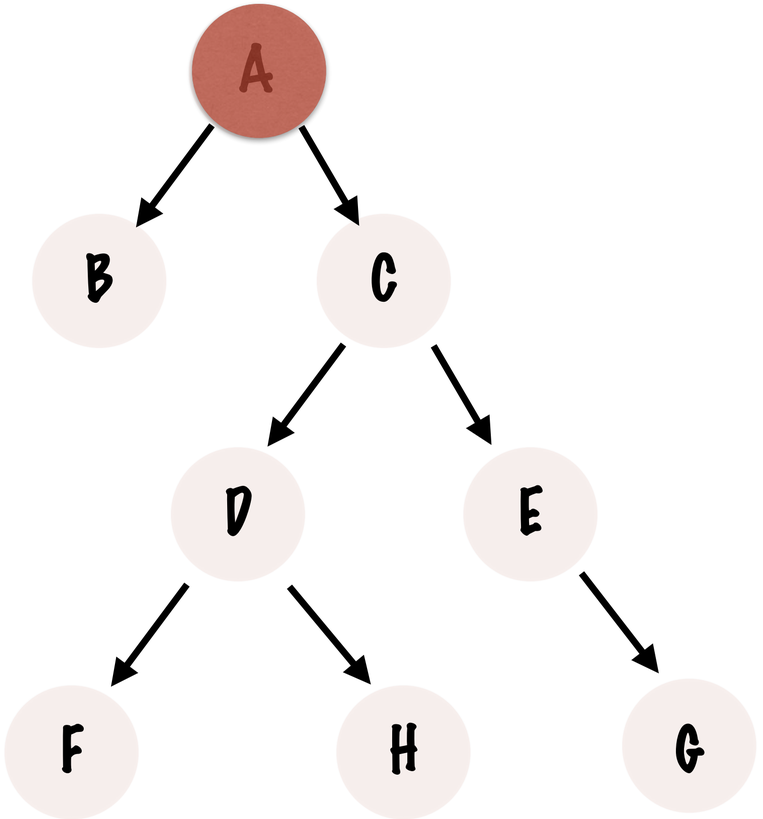
...

↓

HIGHEST LEVEL

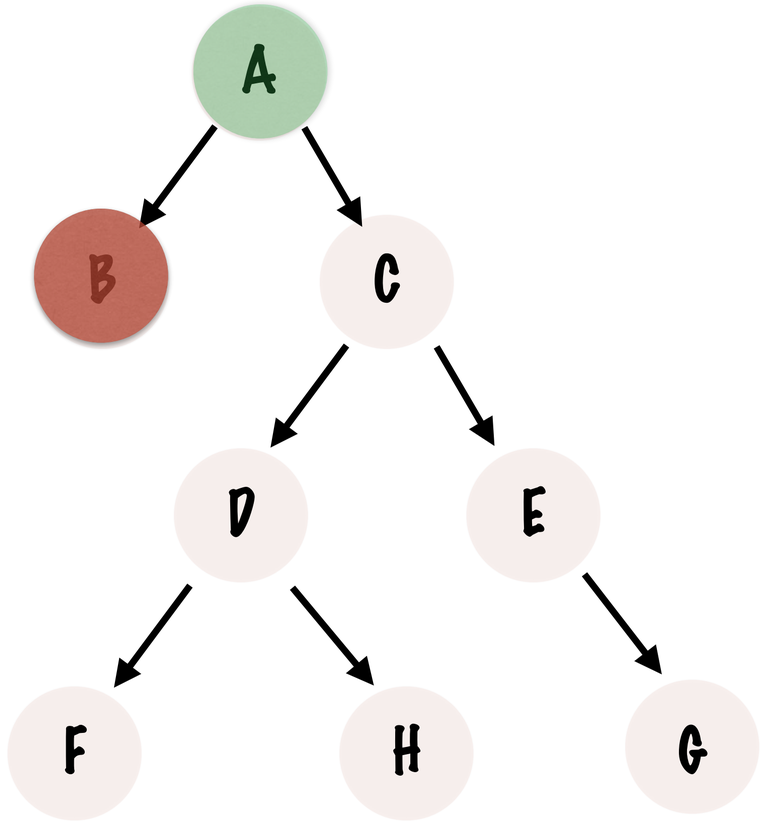
BREADTH-FIRST TRAVERSAL

START VISITING NODES -
STARTING AT THE ROOT



BREADTH-FIRST TRAVERSAL

MOVE FROM LEFT TO RIGHT FOR NODES AT THE SAME LEVEL - B IS THE FIRST NODE TO VISIT AT LEVEL 1

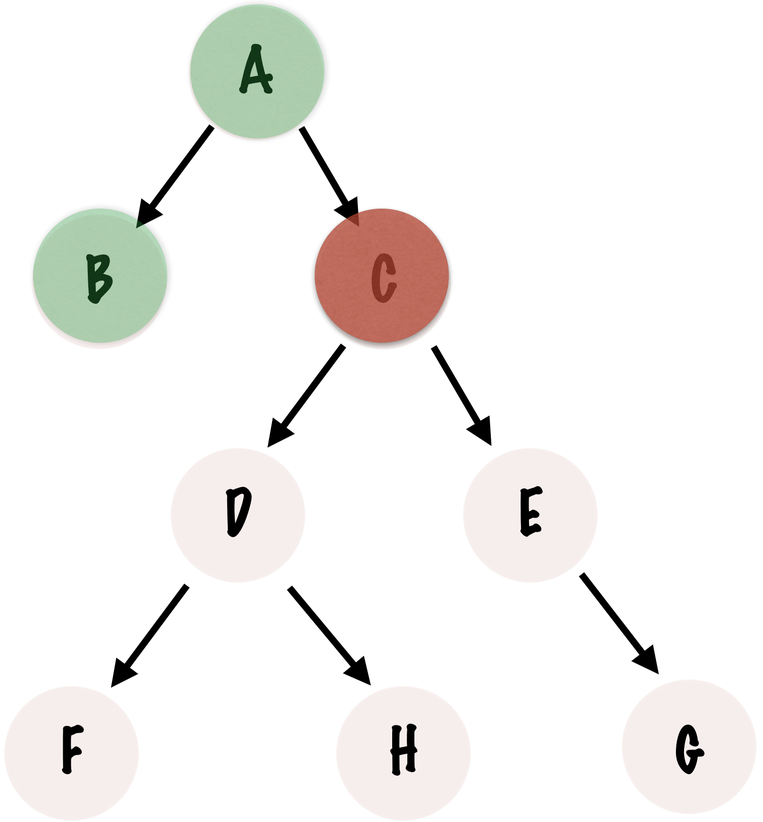


THE NEXT NODE WE VISIT HAS TO BE AT THE SAME LEVEL.

A

BREADTH-FIRST TRAVERSAL

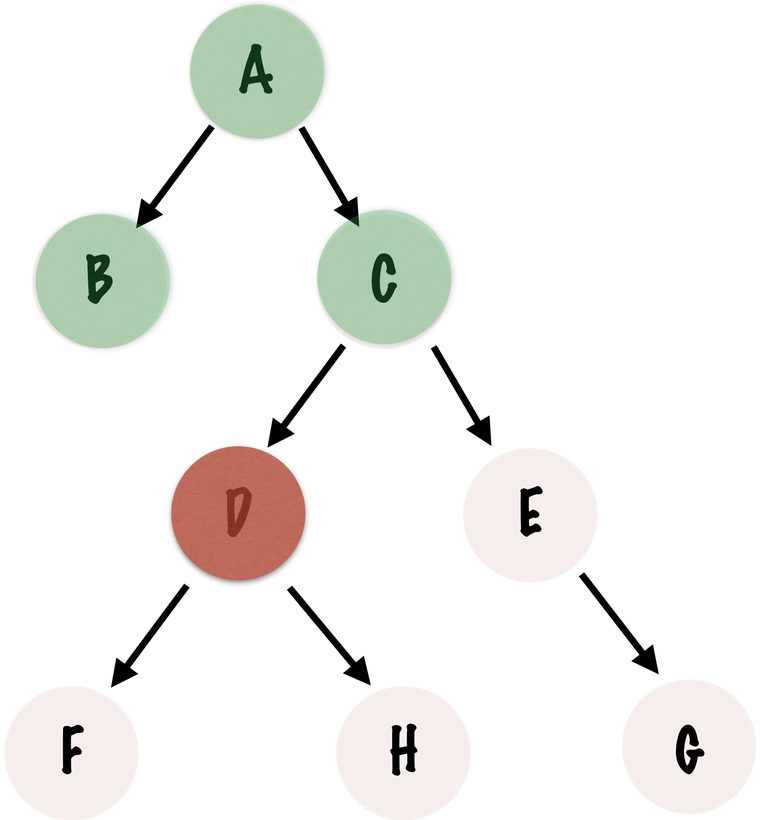
C IS AT LEVEL 1, THERE ARE NO OTHER NODES AT LEVEL 1 - WE CAN NOW MOVE ON TO THE NEXT LEVEL



A->B

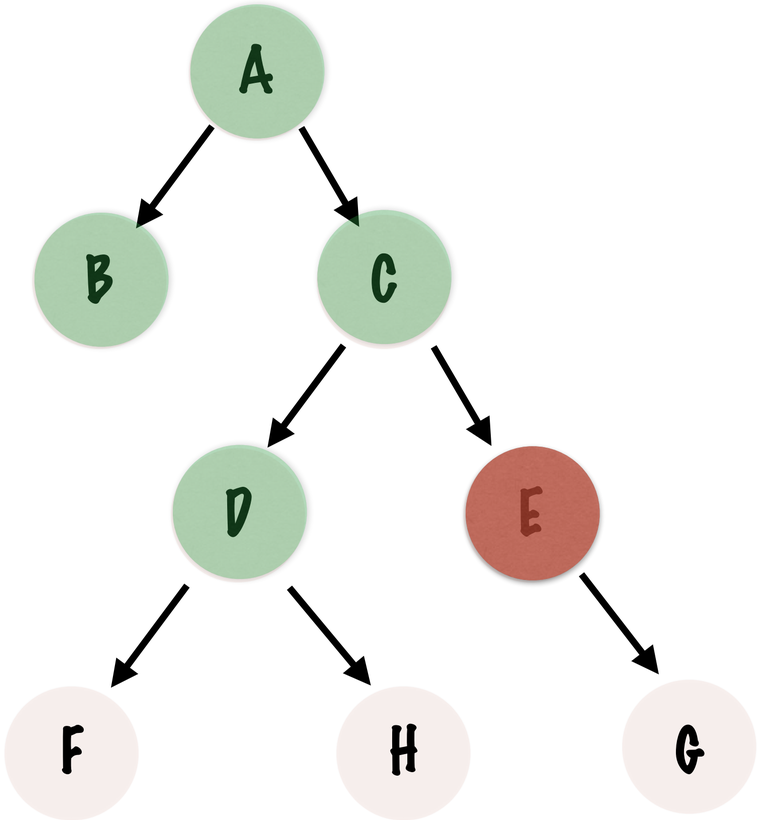
BREADTH-FIRST TRAVERSAL

D IS AT LEVEL 2 AND AS THE LEFT MOST NODE IS VISITED FIRST



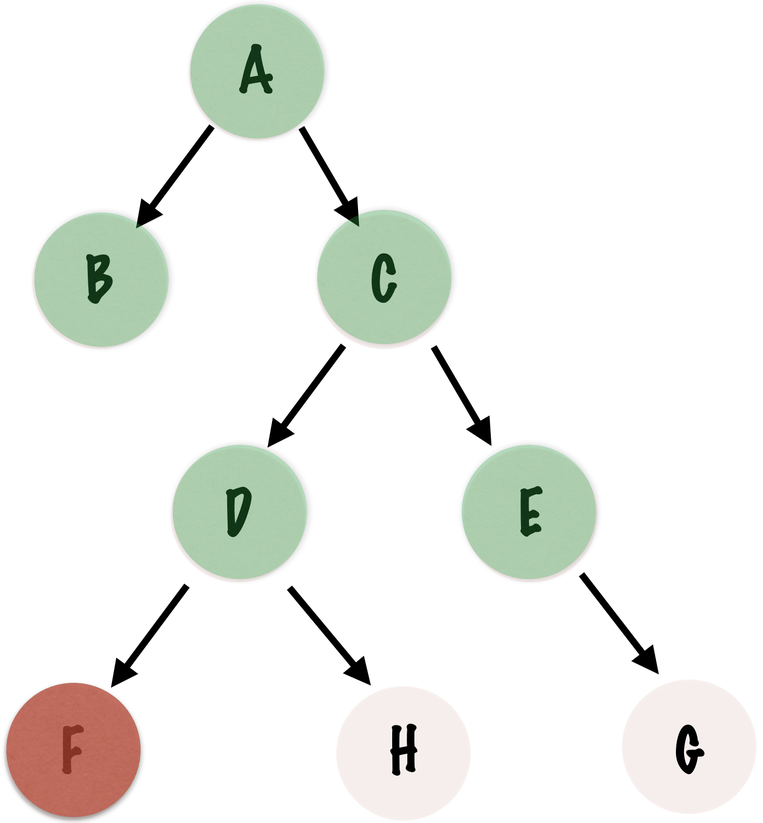
A->B->C

BREADTH-FIRST TRAVERSAL



A->B->C->D

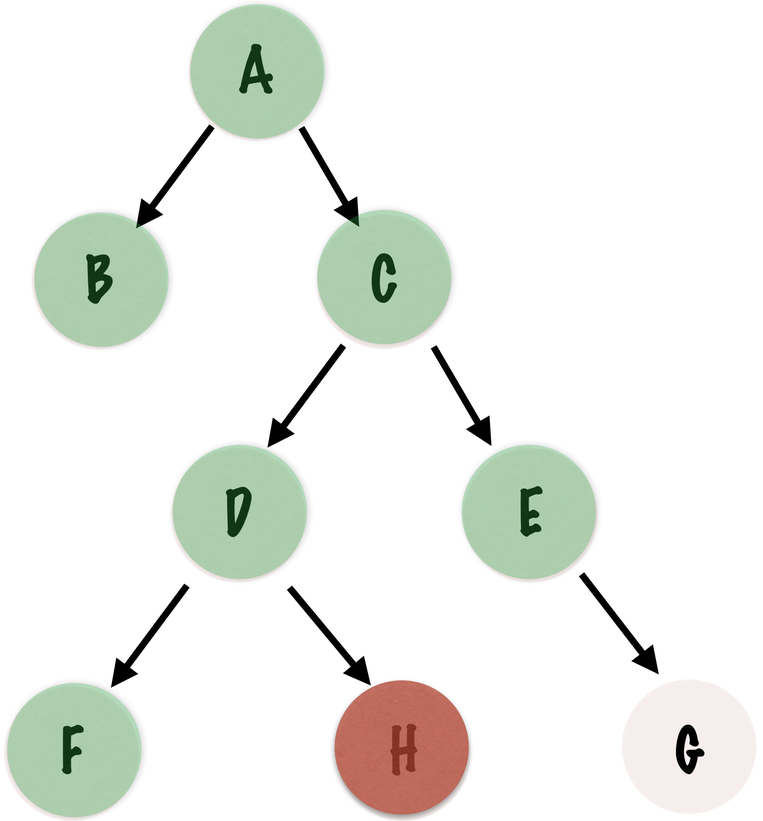
BREADTH-FIRST TRAVERSAL



A->B->C->D->E

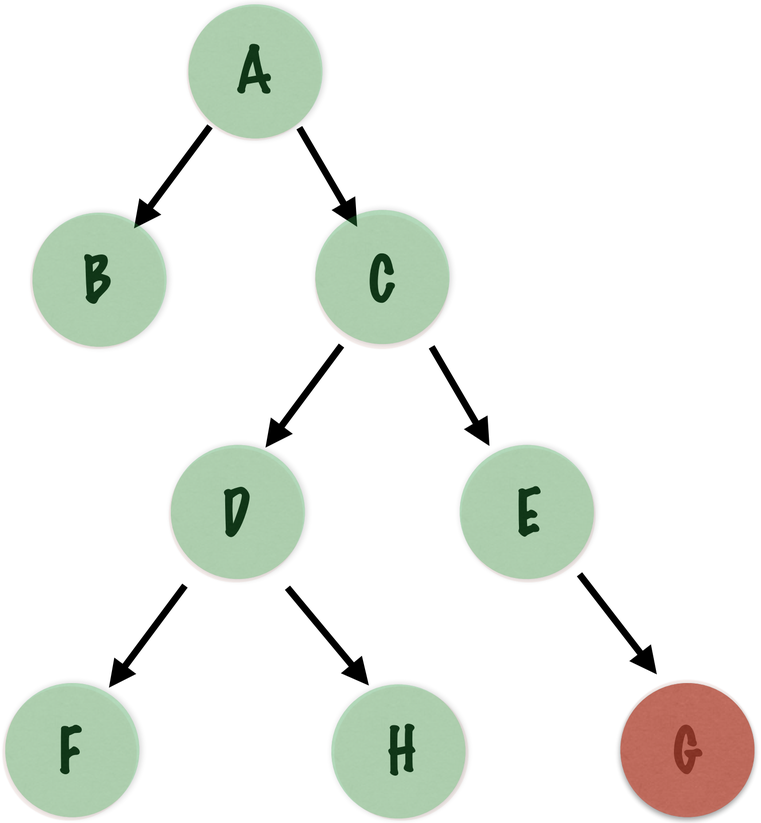
BREADTH-FIRST TRAVERSAL

NOW THE NEXT ELEMENT AT THE SAME LEVEL IS G, THE RIGHT CHILD OF E - THAT IS THE NODE WE VISIT NEXT



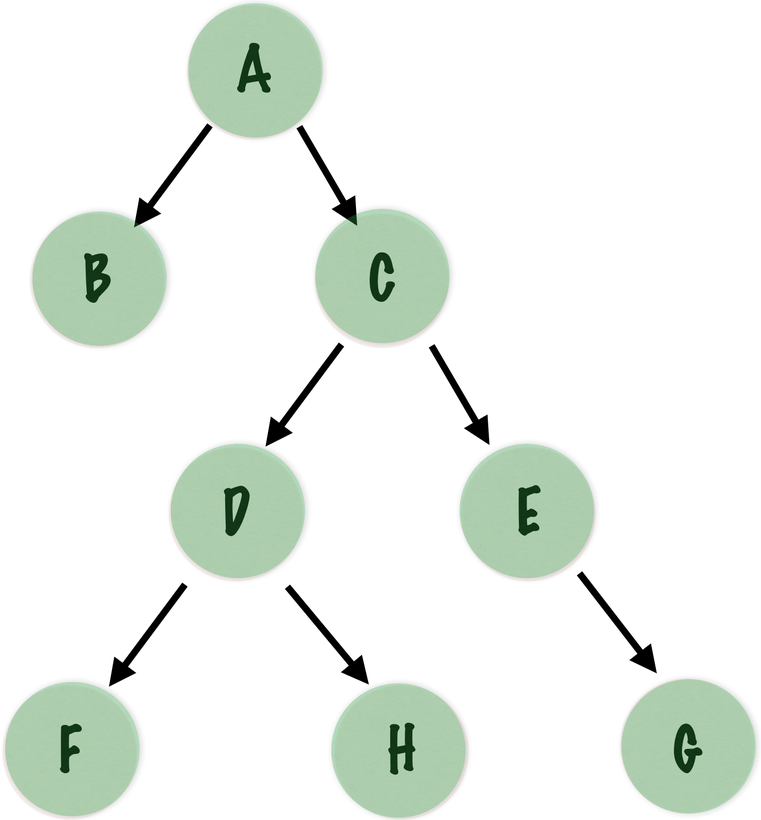
A->B->C->D->E->F

BREADTH-FIRST TRAVERSAL



A->B->C->D->E->F->H

BREADTH-FIRST TRAVERSAL



A->B->C->D->E->F->H->G

IMPLEMENTING BREADTH-FIRST TRAVERSAL

START FROM THE ROOT AND ADD IT TO A QUEUE

YES, THIS USES SOME OF THE DATA STRUCTURES WE'RE ALREADY FAMILIAR WITH!

DEQUEUE AND PROCESS THE NODE

ADD IT'S LEFT AND THEN RIGHT CHILD TO THE QUEUE

CONTINUE THIS AS LONG AS THE QUEUE IS NOT EMPTY

THE NODES GET ADDED LEVEL-WISE FROM LEFT TO RIGHT TO THE QUEUE

AND ARE DEQUEUED AND PROCESSED IN THAT ORDER

BREADTH-FIRST TRAVERSAL

```
public static void breadthFirst(Node root) throws
        Queue.QueueUnderflowException, Queue.QueueOverflowException {
    if (root == null) {
        return;
    }

    Queue<Node> queue = new Queue<>(Node.class);
    queue.enqueue(root);
    while (!queue.isEmpty()) {
        Node node = queue.dequeue();
        print(node);

        if (node.getLeftChild() != null) {
            queue.enqueue(node.getLeftChild());
        }
        if (node.getRightChild() != null) {
            queue.enqueue(node.getRightChild());
        }
    }
}
```

NULL ROOT INDICATES NOTHING TO TRAVERSE

SET UP A QUEUE AND START BY ENQUEUING THE ROOT NODE

AS LONG AS THE QUEUE IS NOT EMPTY, PROCESS THE NODE AT THE HEAD OF THE QUEUE

AND ADD IT'S LEFT AND RIGHT CHILD TO THE QUEUE

ADDING THE LEFT CHILD FIRST ENSURES THAT THE NODES AT THE SAME LEVEL AND PROCESSED FROM LEFT TO RIGHT