DEPTH-FIRST TRAVERSAL

DEPTH-FIRST TRAVERSAL INVOLVES GOING RIGHT TO THE LEAF OF THE BINARY TREE FIRST BEFORE MOVING UP THE TREE

GOING DEEP BEFORE MOVING UP

HERE THERE ARE A WHOLE VARIETY OF POSSIBILITIES IN HOW THE NODES ARE PROCESSED

DEPTH-FIRST TRAVERSALS CAN BE:

PRE-ORDER

IN-ORDER

POST-ORDER

DEPTH FIRST TRAVERSAL

ALL DEPTH FIRST TRAVERSAL ARE MOST EFFICIENTLY AND INTUITIVELY IMPLEMENTED USING RECURSION

AT EVERY POINT WE WORK WITH A SUBTREE ROOTED AT SOME NODE

THE RECURSIVE STEP IS ON 2 SUBTREES - THE LEFT AND THE RIGHT

THE BASE CASE IS WHEN THE ROOT IS NULL

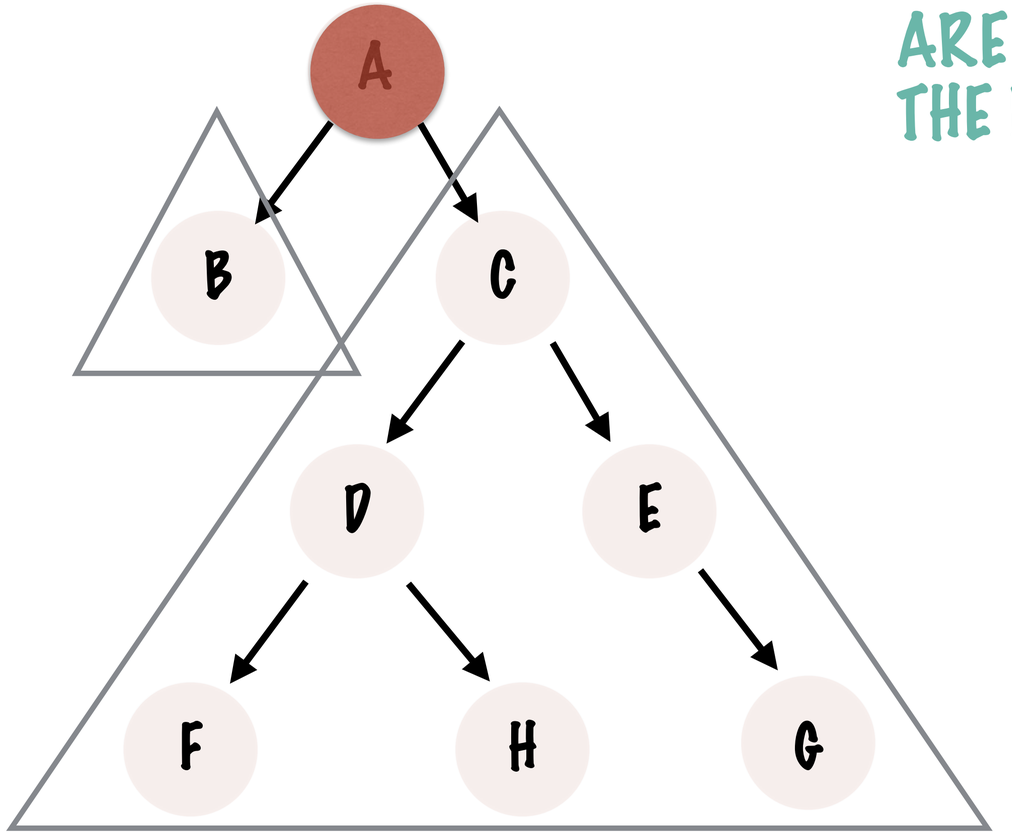
THE PROCESSING CAN BE PERFORMED:

1. BEFORE PRE-ORDER
2. IN-BETWEEN OR IN-ORDER
3. AFTER POST-ORDER

THE RECURSIVE CASE

PRE-ORDER TRAVERSAL

EACH NODE IS PROCESSED FIRST (PRE) BEFORE IT'S RIGHT AND LEFT SUBTREES



THE LEFT SUB-TREES ARE PROCESSED BEFORE THE RIGHT SUBTREES

NODE

↓

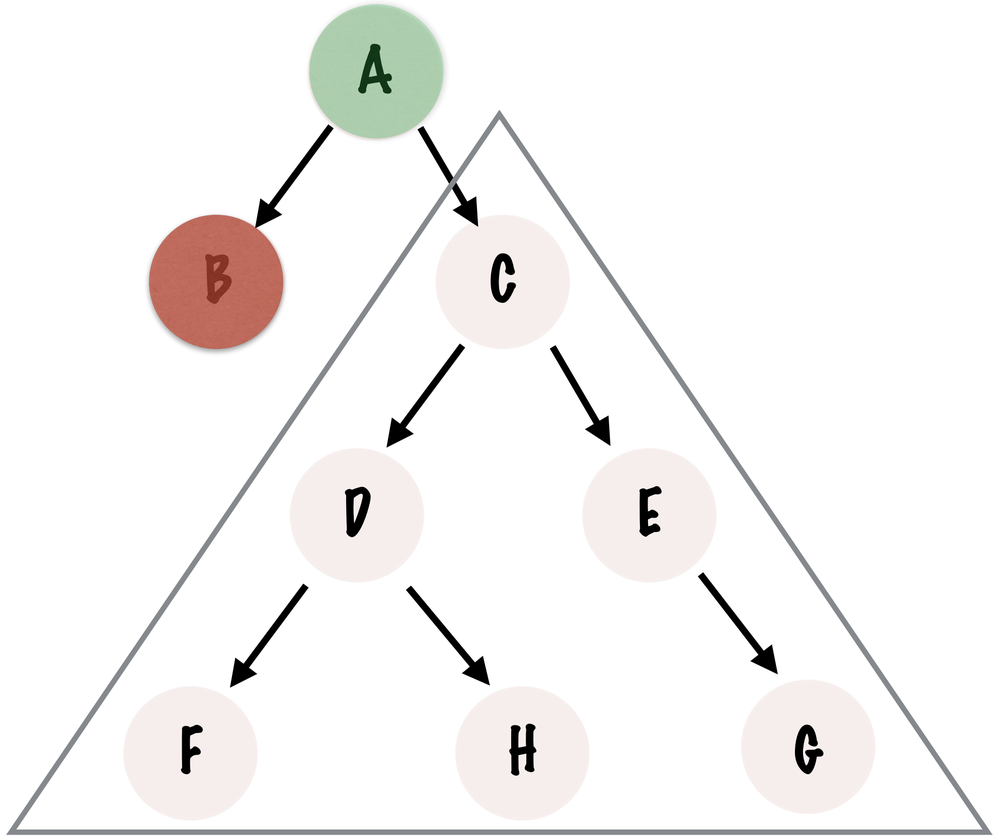
LEFT SUBTREE

↓

RIGHT SUBTREE

A IS PROCESSED BEFORE IT'S SUBTREES

PRE-ORDER TRAVERSAL

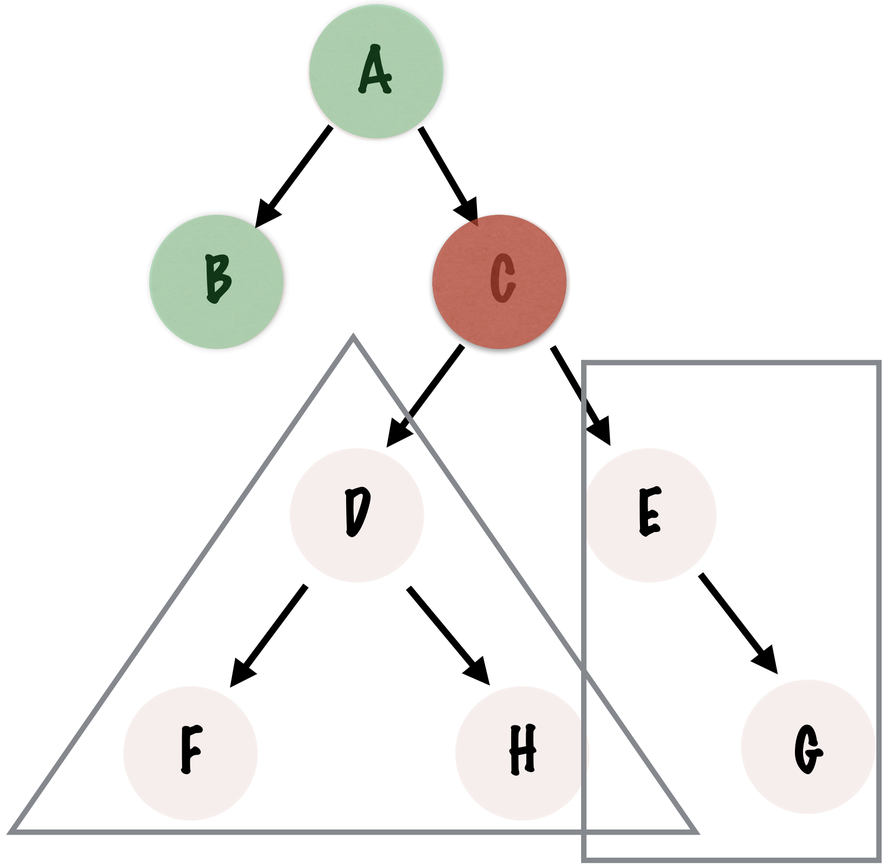


B IS THE LEFT CHILD AND THE ROOT OF THE LEFT SUBTREE. - B IS PROCESSED NEXT

A

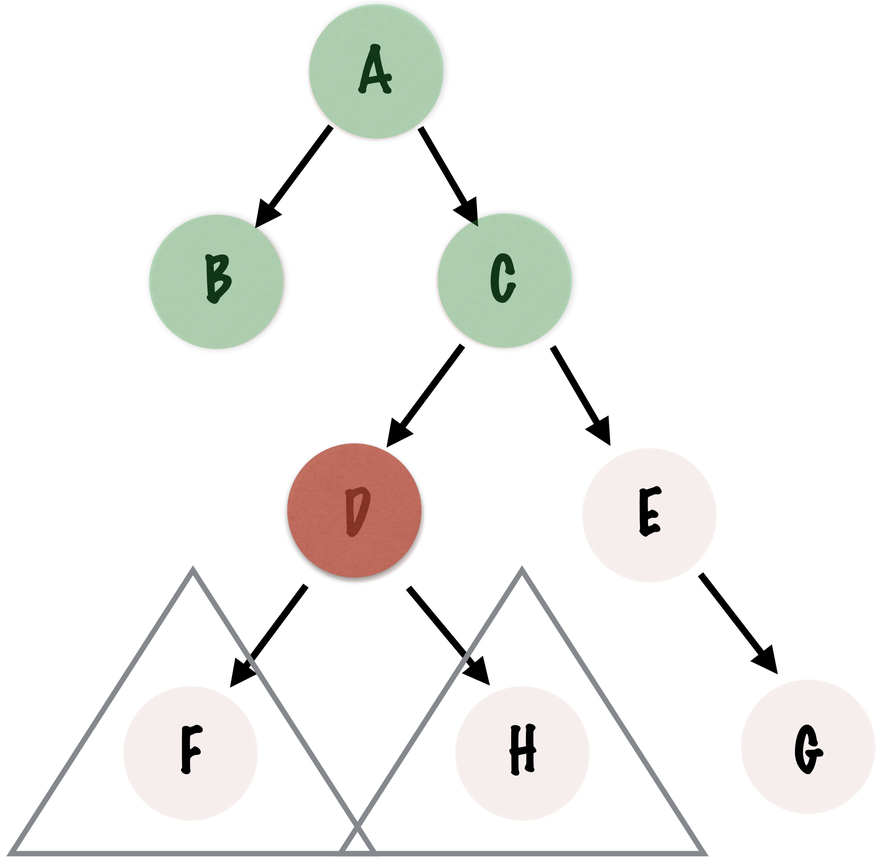
PRE-ORDER TRAVERSAL

C IS THE RIGHT CHILD AND THE ROOT OF THE RIGHT SUBTREE. - C IS PROCESSED NEXT, BEFORE IT'S SUBTREES



A->B

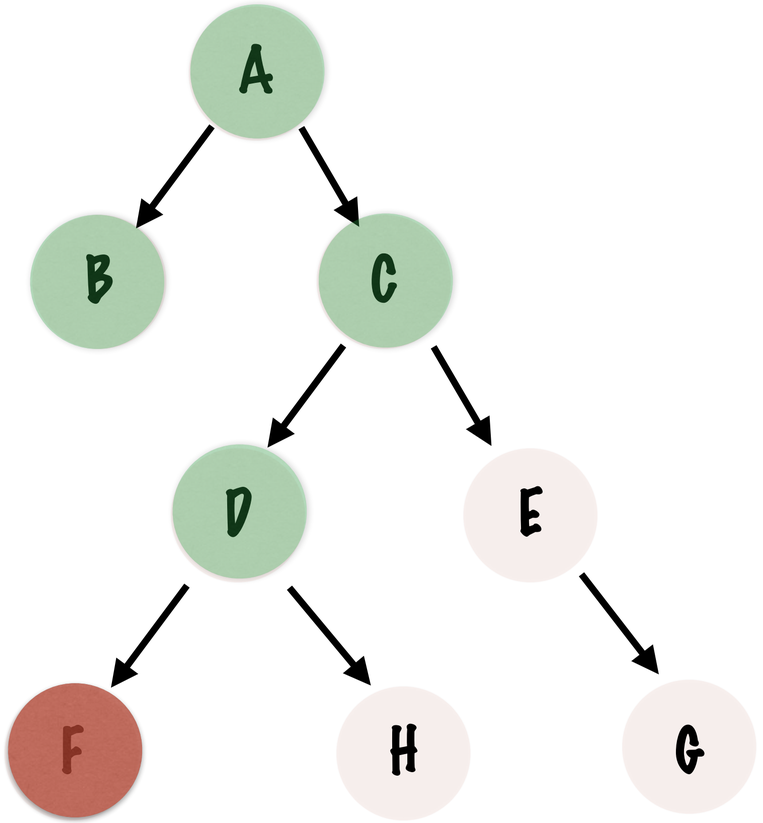
PRE-ORDER TRAVERSAL



D IS THE LEFT CHILD AND
IS PROCESSED NEXT

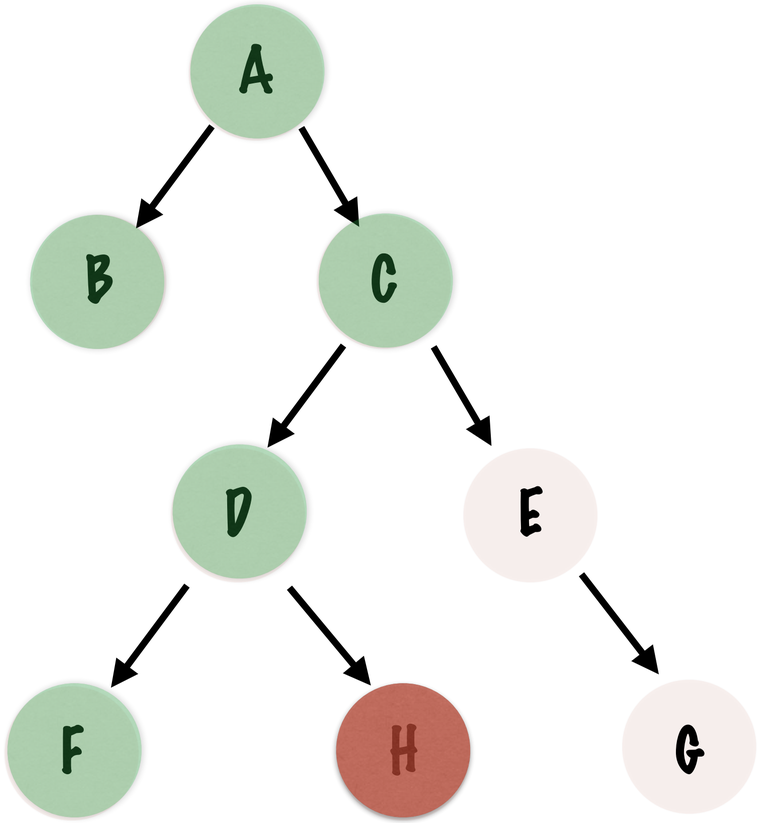
A->B->C

PRE-ORDER TRAVERSAL



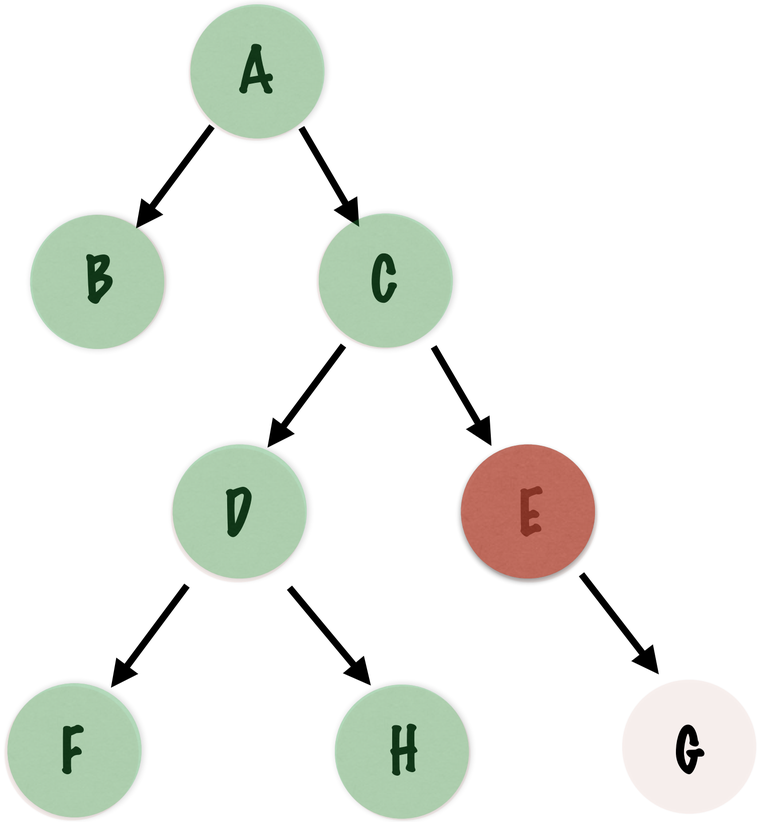
A->B->C->D

PRE-ORDER TRAVERSAL



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

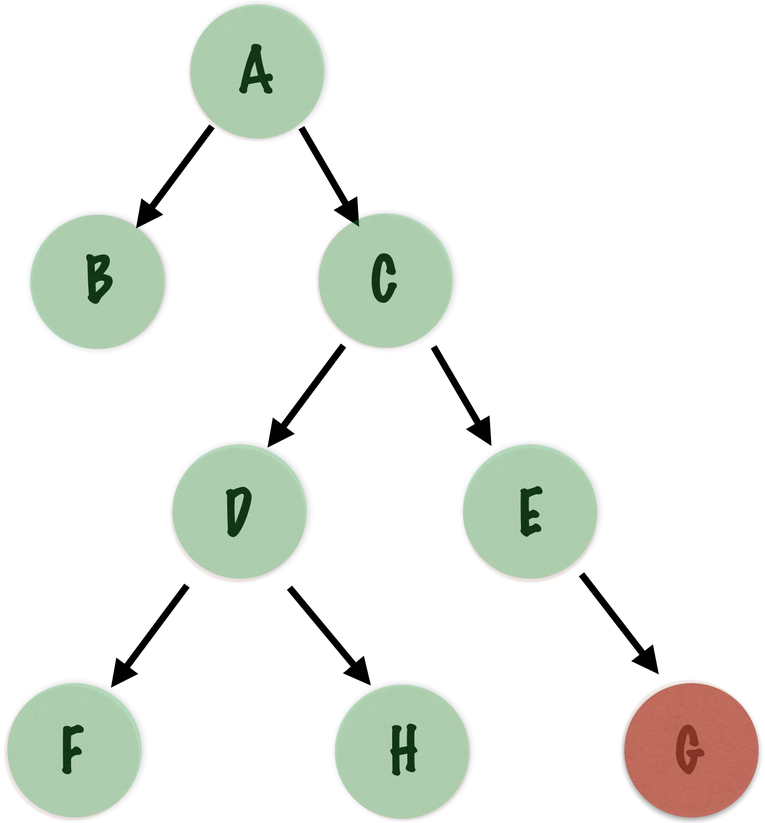
PRE-ORDER TRAVERSAL



WE'RE DONE WITH THIS SUBTREE, MOVE ON TO THE RIGHT CHILD OF C

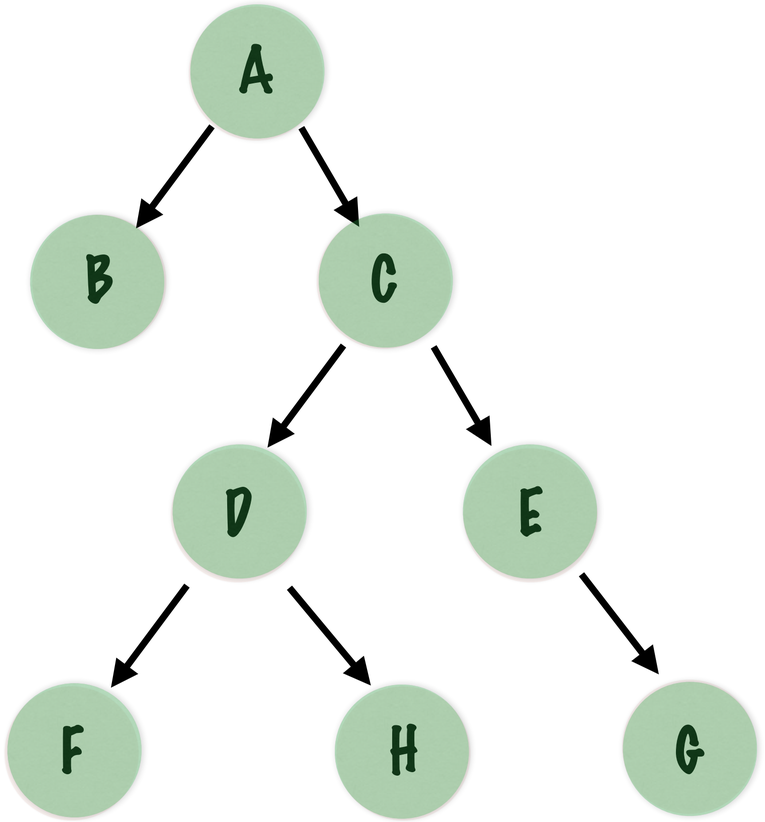
A->B->C->D->F->H

PRE-ORDER TRAVERSAL



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

PRE-ORDER TRAVERSAL



ALL NODES HAVE BEEN VISITED!

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

PRE-ORDER TRAVERSAL CODE

```
public static void preOrder(Node<Character>root) {
    if (root == null) {
        return;
    }

    print(root);
    preOrder(root.getLeftChild());
    preOrder(root.getRightChild());
}
```

BASE CASE - NOTHING TO TRAVERSE

PROCESS THE NODE BEFORE RECURSING TO THE LEFT AND RIGHT SUBTREES