Depth First Search

Implemented Using **Stack**, operating in **LIFO**

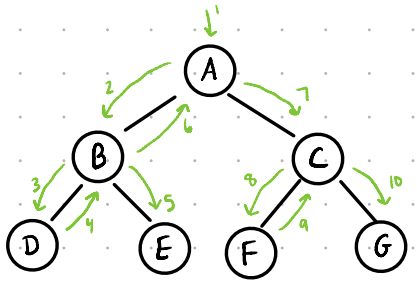
Explores as deep as possible before backtracking

Applications: Tree traversal, pathfinding in mazes.

Different Ways: Pre-Order, In-Order, Post-Order

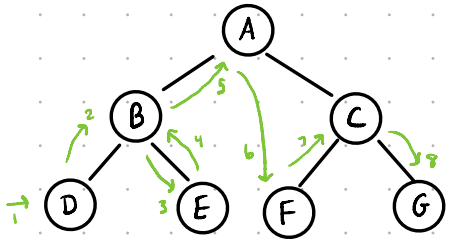
Pre-Order

Visit the node first, then it's left child, and finally its right child



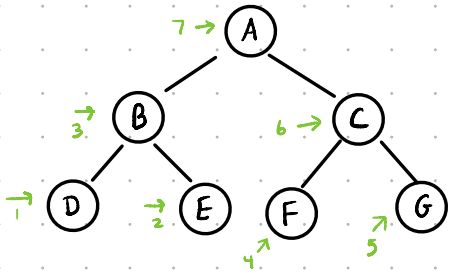
In-Order

visit the left child first, then the node, and finally the right child



Post-Order

visit the left child, then the right child, and finally the node



Can use Recursion or Iteration.

Recursion

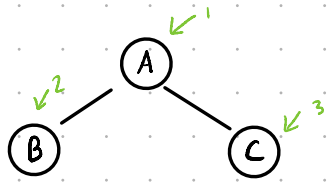
using Call Stack

slower and uses call stack built in PL which is more limited in size

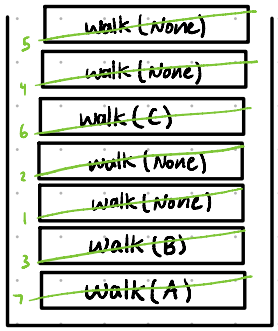
Pre-Order

b/c print statement is above the two recursive calls to the children of A

* when function is called it's pushed on call stack and popped when it returns




```
def walk(tree):
    if tree is not None:
        print(tree)
        walk(tree.left)
        walk(tree.right)
```



→ B has no right child so returns None

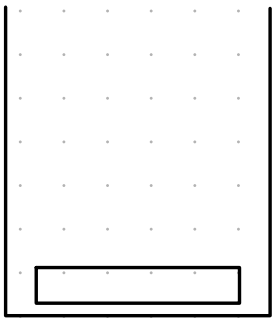
→ B has no left child so returns None

→ walk(tree.left)

* order of popping

In-Order

```
def walk(tree):
    if tree is not None:
        walk(tree.left)
        print(tree)
        walk(tree.right)
```



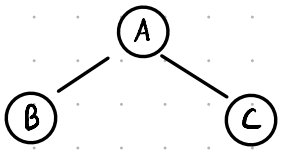
Post-Order

```
def walk(tree):
    if tree is not None:
        walk(tree.left)
        walk(tree.right)
        print(tree)
```

Iterative

Using your own explicit stack

faster and doesn't have same limitations as using the call stack




```
def walk(tree, stack):
    stack.append(tree)
    while len(stack) > 0:
        node = stack.pop()
        if node is not None:
            print(node)
            stack.append(node.right)
            stack.append(node.left)
```