Section 5: Tree Traversals.md

Section 5: Tree Traversals

Raymond Guo

Disclaimer: I'm a little rusty in 61B and actually relearning with y'all :). These are just my personal notes, so there may be errors in this document; if so, let me know! Also, this isn't affilianted with CSM or any Berkeley organization :).

Tree Traversals (Q3)

There are two main types of tree traversals:

Level Order Traversal / Breadth First Traversal

In level order traversal, we visit the nodes of the tree starting from the root, in the order of distance from the root. On an exam, you can draw out the tree levels and print out the nodes in the order from top level to bottom level:

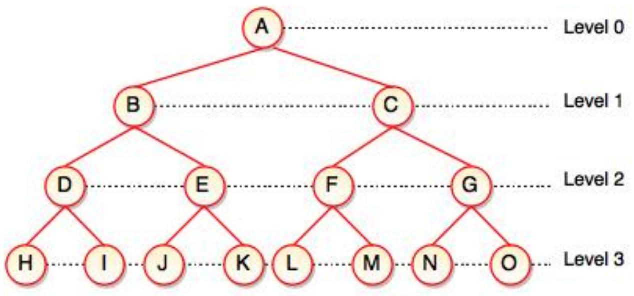


Fig. Complete Binary Tree

The level order traversal of this tree is `ABCDEFGHIJKLMNO` .

Depth First Traversal

In depth first traversal, we visit the nodes of the tree starting from the root, going to the deepest node possible (until we reach a leaf node) before backtracking and going to the next child. There are three main types of depth first traversals:

- Preorder: If we had a binary tree, we would first print the node's value, then visit the left child, then visit the right child. The preorder traversal of the above tree is `ABDHIEJKCFLMGNO` .
- Postorder: If we had a binary tree, we would first visit the left child, then visit the right child, then print the node's value. The postorder traversal of the above tree is `HIDJKEBLMFNOGCA` .
- Inorder: If we had a binary tree, we would first visit the left child, then print the node's value, then visit the right child. The inorder traversal of the above tree is `HDIBJEKALFMCNGO` .

There's a pretty simple trick for getting the preorder, postorder, and inorder traversals. First, draw a shape around the tree, then traverse through the shape's perimeter in a counter-clockwise direction. You'll notice that when you make one cycle, you visit each node 3 times (one time when you're passing the left side of the node, one time when you're passing the bottom of the node, and one time when you're passing the right side of the node). When you want preorder, print out the node's value when you pass by the left side of the node. When you want inorder, print out the node's value when you pass by the bottom of the node. When you want postorder, print out the node's value when you pass by the right side of the node. For a better (and more visual) explanation of this trick, see https://www.youtube.com/watch?v=WLvU5EQVZqY.

Implementing Traversals

There are two main types of ways to implement DFT and BFT: iteratively and recursively. For each of the types of DFT/BFT, one method is typically easier to implement than the other. Here, I'm just going to outline the easiest ways to implement *some* of the traversals using each of these two methods.

Iteratively

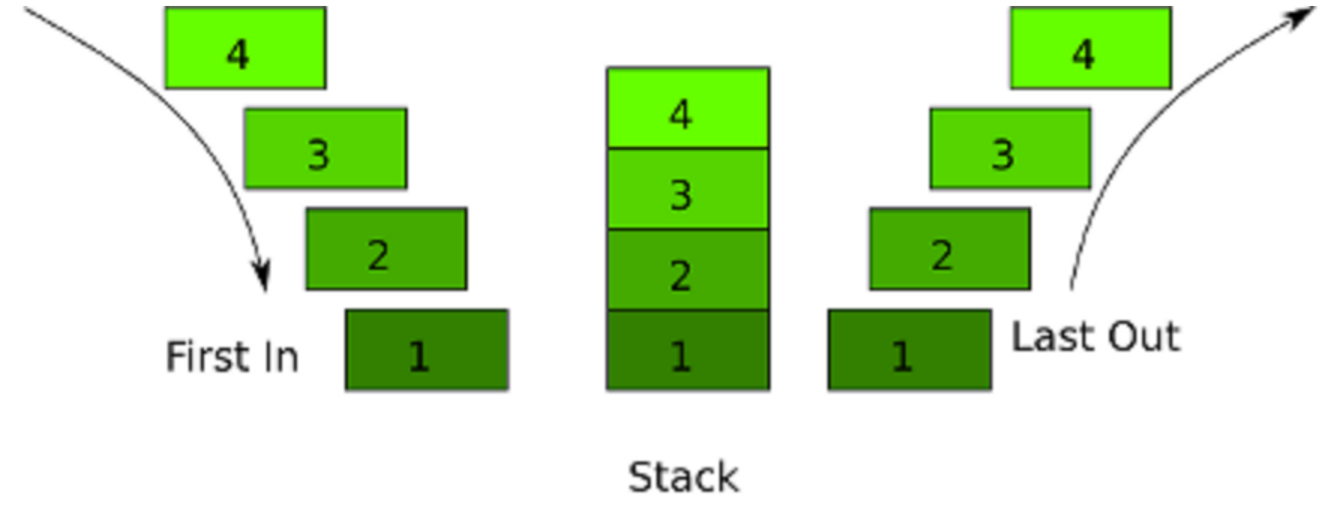
You can very easily implement breadth first traversal and preorder depth first traversal using an iterative algorithm. Before that, though, let's first introduce the general code that an iterative traversal implementation follows:

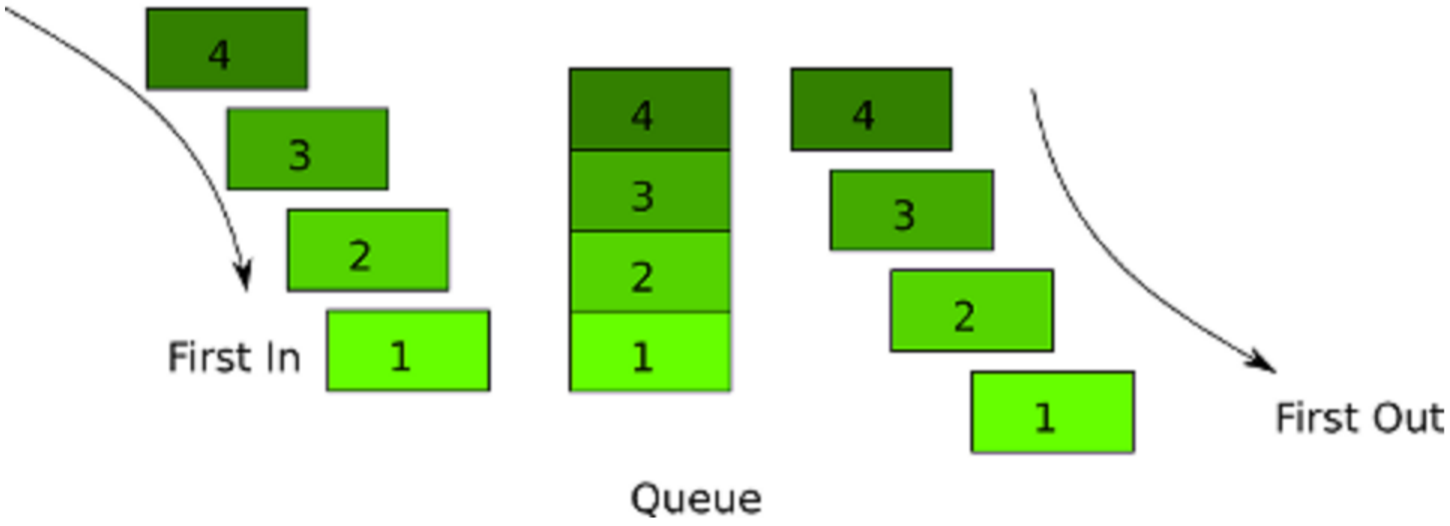
```
public void treeTraversal(Fringe<Node> fringe) {
    fringe.add(root);
    while (!fringe.isEmpty()) {
        Node node = fringe.remove();
        System.out.print(node.value);
        if (node.left != null) {
            fringe.add(node.left);
        }
        if (node.right != null) {
            fringe.add(node.right);
        }
    }
}
```

If you trace through this algorithm, it essentially repeatedly removes a single node from the fringe, prints out its value, then adds its children to the fringe. As a result, the fringe, in this case, is essentially a set of nodes that are on the "to-visit" list.

At times, the fringe will have multiple nodes. How do we know which order we should remove nodes from the fringe? Here is where we introduce queues and stacks:

Both queues and stacks are lists, but the order that you remove from is different. I imagine queues as a one-way line. The node you remove from a queue is the one that was most recently added (First In, First Out). Stacks, on the other hand, is like a pile of plates; the node you remove from a stack is the one that was most recently added (First In, Last Out; or Last In, First Out). Here's a visualization of the two:





Here is the crucial part. If you trace through the algorithm above:

> If the fringe is a queue, then you get breadth first traversal. If the fringe is a stack, then you get preorder depth first traversal.

Recursively

You can very easily implement all 3 types of depth first traversal using an recursive algorithm:

```
public void preorder(Node node) {
    if (node == null) {
        return;
    }
    System.out.println(curr.node);
    preorder(node.left);
    preorder(node.right);
}

public void inorder(Node node) {
    if (node == null) {
        return;
    }
    inorder(node.left);
    System.out.println(curr.node);
    inorder(node.right);
}

public void postorder(Node node) {
    if (node == null) {
        return;
    }
    postorder(node.left);
    postorder(node.right);
    System.out.println(curr.node);
}
```