CS 260
Binary Tree Traversals

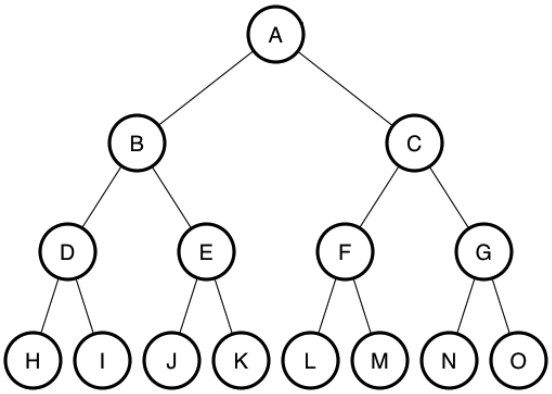
Traversals

In data structures, a traversal is an algorithm that visits all elements of the structure. Examples of this are a linear traversal of an array (starting at lowest index and ending at the highest index) or a head to tail traversal of a linked list. This document covers the three common traversal algorithms of binary trees.

Note that these are described as recursive functions. It is relatively trivial to implement them as iterative functions using an explicit stack, but such code is more complex. There are several algorithms for fully iterative solutions, but they are not covered in this course. Knowing the algorithms as presented here is considered basic computer science knowledge and is necessary for success in this course.

Example Tree

Consider the following tree containing the letters A to O.



Three Traversals

While the nodes could be visited in alphabetical order, that appears to be hard to program. Visit A, then go to left child B, then back to parent A, then right child C, the back to parent A, back to left child B, and so on. Instead of walking up and down the tree in this manner, the following three algorithms use recursion to make it simple.

The three algorithms are pre-order, in-order, and post-order, where each is defined by how it treats a node and its two subtrees:

- pre-order visits the node, then the left subtree, then the right subtree
- in-order visits the left subtree, then the node, then the right subtree
- post-order visits the left subtree, then the right subtree, then the node

The names come from where in the process of visiting the subtrees, the node itself is visited — for pre-order it comes first, in-order it is in the middle, and post-order it comes last. Considering these as recursive methods, the pseudo code is as follows (assuming returning a string containing the nodes as visited). Again, the pseudocode uses nullptr to stand in for nullptr/None/NULL.

Recursive Algorithms

```
recPreOrder(ptr)
    buffer = ""

    // if done with branch, return empty
    if ptr == nullptr
        return buffer

    // build buffer in proper order
    buffer += (to_string(ptr.value) + " ")
    buffer += recPreOrder(ptr.left)
    buffer += recPreOrder(ptr.right)

    return buffer


recInOrder(ptr)
    buffer = ""

    // if done with branch, return empty
    if ptr == nullptr
        return buffer

    // build buffer in proper order
    buffer += recInOrder(ptr.left)
    buffer += (to_string(ptr.value) + " ")
    buffer += recInOrder(ptr.right)

    return buffer


recPostOrder(ptr)
    buffer = ""

    // if done with branch, return empty
    if ptr == nullptr
        return buffer

    // build buffer in proper order
    buffer += recPostOrder(ptr.left);
    buffer += recPostOrder(ptr.right);
    buffer += (to_string(ptr.value) + " ")

    return buffer
```