ECE 250 *Algorithms and Data Structure*
Department of Electrical and Computer Engineering
University of Waterloo
Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Notes 6.3.
Assistances and comments will be acknowledged.

6.3 In-order traversals

In this topic, we will introduce in-order depth-first traversals, we will see that in-order traversals apply to binary trees but not to general or *N*-ary trees with $N > 2$, and then we will introduce *M*-way trees which allow for multiple children but also allow for the storage of linearly ordered data (unlike *N*-way trees) and in-order traversals.

6.3.1 In-order depth-first traversals

We have discussed both pre-order and post-order depth-first traversals for a general tree. A binary tree, however, allows a third option: visiting the node between the visit to the left and right sub-trees (if any). This is possible because we have a distinction between the left and right sub-trees.

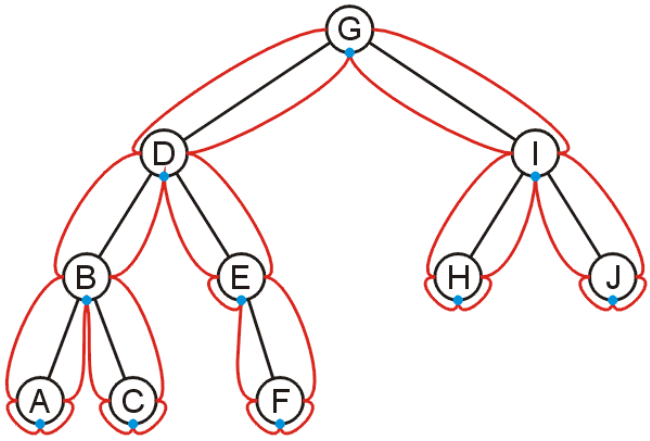


Figure 1. An in-order traversal of a binary tree.

Thus, an in-order depth-first traversal is made on the left sub-tree, when that completes, the current node is visited, and then an in-order traversal is made on the right sub-tree. This would produce the values

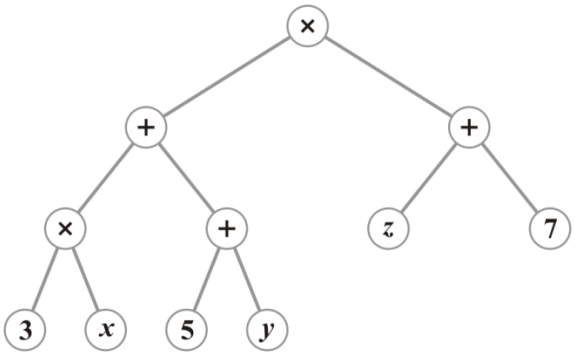
ABCDEFGHIJ

as opposed to the sequence GDBACEFIHJ for a pre-order depth-first traversal and ACBFEDHJIG for a post-order depth-first traversal.

Note also that in a binary search tree, an in-order traversal visits the nodes in the linear order of the elements. We will look at an alternate example of an application using in-order traversals.

6.3.1.1 In-order traversals of expression trees

Consider the expression tree shown in



An in-order traversal produces $3 \times x + 5 + y \times z + 7$; however, what is meant by this expression is

$$(3x + 5 + y)(z + 7)$$

Upon inspection, it would appear if addition appears as a child of multiplication, parenthesis must be placed around the sum being printed. In this case, both the left and right sub-trees of the root satisfy this condition. Thus, an in-order traversal may be implemented as follows:

```
class Algebraic;

void pretty_print( Algebraic *parent ) {
    if ( !leaf() ) {
        if ( parent->precedence() > precedence() ) {
            cout << "(";
        }  // pre-order operation

        // traverse the left sub-tree
        left_tree->pretty_print( this );
    }

    // only print a 'x' if both operands are numeric
    if ( is_multiplication() ) {
        if ( left_tree->numeric() && right_tree->numeric() ) {
            cout << " x ";
        }
    } else {
        cout << this;  // print this object
    }

    if ( !leaf() ) {
        right_tree->pretty_print( this ); // traverse right sub-tree

        if ( parent->precedence() > precedence() ) {
            cout << ")";
        }  // post-order operation
    }
```

6.3.3 In-order traversals on general or *N*-ary trees

An in-order depth-first traversal does not make sense with a general tree or an N-ary tree with $N > 2$, as there is no obvious order between the node and its children. The in order traversal only works for binary trees because of the relative order of the left and right sub-trees.