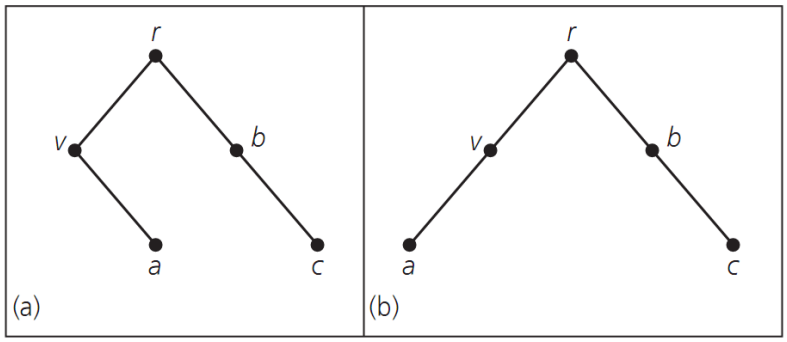
COMP 2121
**DISCRETE MATHEMATICS**

# Lecture 15


## Tree Traversals

A **binary tree** is a rooted tree in which each vertex has at most two children and each child is designated as a **left child** or a **right child**. Thus, in a binary tree, each vertex may have 0, 1 or 2 children. When drawing a binary tree, we will follow customary practice and draw a left child to the left and below its parent and a right child to the right and below its parent. The left subtree of a vertex V in a binary tree is the graph formed by the left child L of V, the decedents of L, and the edges connecting these vertices. The right subtree of V is defined in an analogous manner.

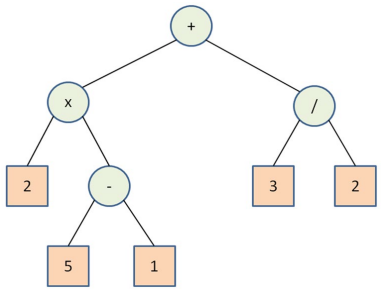
In computer science, a binary tree is a tree data structure in which each node has at most two children. In part (a) of the figure below, the vertex v has a right child a, whereas in part (b) vertex a is the left child of v. Consequently, these trees are not viewed as the same tree.




Binary trees are used extensively in computer science to organize data and describe algorithms. For example, during the execution of a computer program, it may be necessary to evaluate arithmetic expressions such as:

$$2 \times (5 - 1) + 3 / 2$$

Our knowledge of conventions for the order of operations tells us how to proceed with this calculation: Scan from left to right, first doing multiplication and division, and then addition and subtraction, with the understanding that parentheses have priority. An alternative approach is to represent an arithmetic expression by a binary tree and then process the data in some other way.

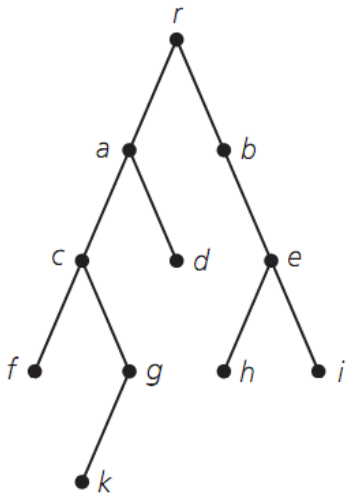



We have seen that arithmetic expression can be represented by an expression tree. Now we must process the expression tree in some way so as to obtain an evaluation of the original expression. We are looking for a systematic way to examine each vertex in the expression tree exactly once.

Postorder Traversal Algorithm

Step 1: (start)
Go to the root.
Step 2: (go left)
Go to the left subtree, if one exists, and do a postorder traversal.
Step 3: (go right)
Go to the right subtree, if one exists, and do a postorder traversal.
Step 4: (visit)
Visit the root.

**Example 1.** Apply the postorder traversal to the following binary rooted tree.

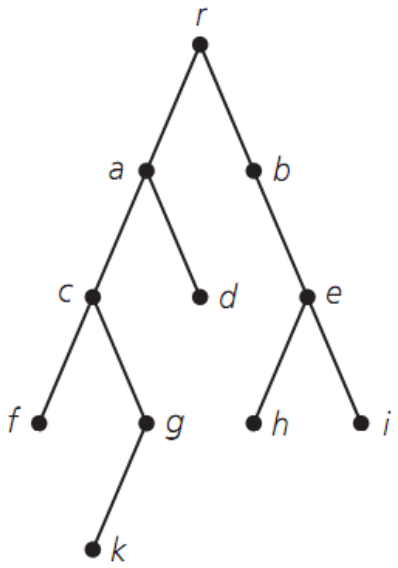



The vertices are visited in the following order:

Inorder Traversal Algorithm

Step 1: (start)
Go to the root.
Step 2: (go left)
Go to the left subtree, if one exists, and do an inorder traversal.
Step 3: (visit)
Visit the root.
Step 4: (go right)
Go to the right subtree, if one exists, and do a inorder traversal.

**Example 2.** Apply the inorder traversal to the following binary rooted tree.




The vertices are visited in the following order:

Binary search tree is a binary tree in which each internal node X stores an element such that:

- the elements stored in the left subtree of X are less than or equal to X, and
- the elements stored in the right subtree of X are greater than X.

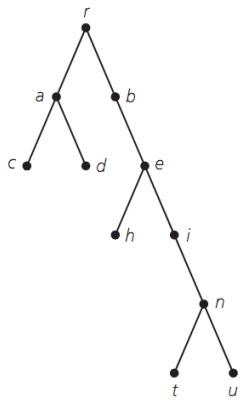
**Example 3.** Build BST for the elements 7, 9, 2, 4, 5.

**Example 4.** Sort the following numbers in increasing order 15, 10, 12, 4, 7, 28, 2, 35, 8, 5, 21.

Preorder Traversal Algorithm

Step 1: (visit)
Visit the root.
Step 2: (go left)
Go to the left subtree, if one exists, and do an preorder traversal.
Step 3: (go right)
Go to the right subtree, if one exists, and do a preorder traversal.

**Example 5.** Apply the preorder traversal to the following binary rooted tree.



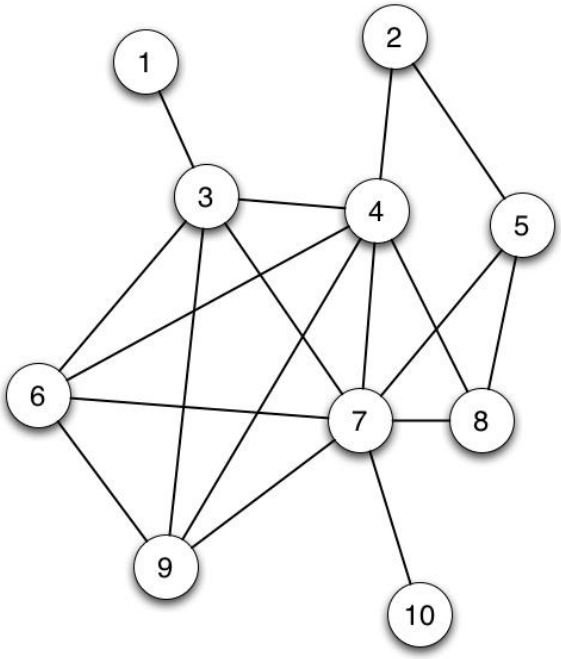

The vertices are visited in the following order:

**Depth First Search** is another way of traversing graphs, which is closely related to preorder traversal of a tree.

DFS Algorithm

```
dfs(vertex v)
   {
   visit(v);
   for each neighbor w of v
         if w is unvisited
         {
         dfs(w);
         add edge vw to tree T
         }
   }
```

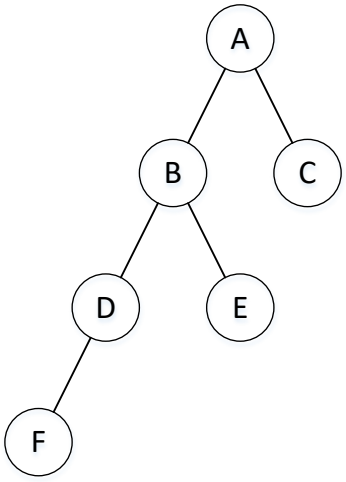
DFS Example:



**Example 6.** Define "SomeOrder" tree Traversal Algorithm as follows:

SomeOrder Traversal Algorithm:
Step 1: (start)
Go to the root.
Step 2: (go **left**)
Go to the left subtree, if one exists, and do a SomeOrder traversal.
Step 3: (go **right**)
Go to the right subtree, if one exists, and do a SomeOrder traversal.
Step 4: (go **left**)
Go to the left subtree, if one exists, and do a SomeOrder traversal.
Step 5: (**visit**)
Visit the root.

For the tree shown below, list the vertices according to the SomeOrder traversal.




SomeOrder: