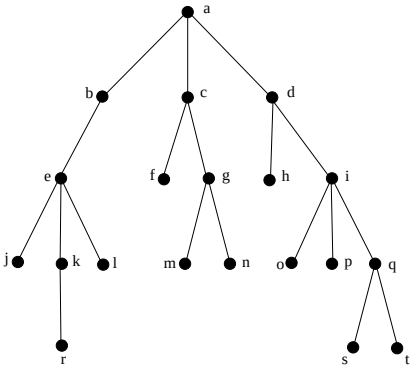
Math 310
Tree Traversal Handout

Main Idea: A **traversal algorithm** is a procedure for visiting every vertex of an ordered rooted tree. The three methods we will look at give instructions for visiting the vertices of am ordered rooted tree using a recursive procedure.

1. **Visiting Vertices in Preorder:**

   Let T be an ordered rooted tree with root r. If T has only one vertex, then r is the *preorder traversal* of the tree T. Otherwise, let $T_1, T_2, ..., T_n$ be the subtrees associated with each child of r ordered from left to right. In *preorder traversal*, we begin by visiting the root r. We then traverse the subtree T_1 using preorder, followed by T_2, and so on, until we finish visiting T_n. Note that each subtree is considered as a separate tree, so when applying preorder to a subtree, we begin by visiting its root, and then applying preorder traversal to each of *its* subtrees. [The mantra here is "root first, then traverse subtrees from left to right"]

   Example: Consider the following tree:



   Applying the procedure described above, the preorder traversal is:

2. **Visiting Vertices in Inorder:**

   Let T be an ordered rooted tree with root r. If T has only one vertex, then r is the *inorder traversal* of the tree T. Otherwise, let $T_1, T_2, ..., T_n$ be the subtrees associated with each child of r ordered from left to right. In *inorder traversal*, we traverse the subtree T_1 using inorder, and then visit the root of T_1, followed by T_2 and then its root, and so on, until we finish visiting T_n, and its root, and then visiting the root r. Note that each subtree is considered as a separate tree, so when applying inorder to a subtree, we apply the procedure for inorder traversal to each of *its* subtrees. [The mantra here is "subtree root last, traverse subtrees from left to right"]

   Example: Applying inorder to the tree shown above gives the following:

3. **Visiting Vertices in Postorder:**

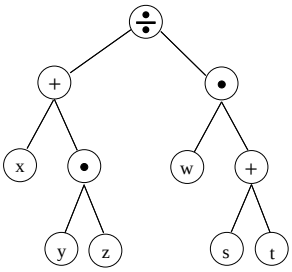
   Let T be an ordered rooted tree with root r. If T has only one vertex, then r is the *postorder traversal* of the tree T. Otherwise, let $T_1, T_2, ..., T_n$ be the subtrees associated with each child of r ordered from left to right. In *postorder traversal*, we traverse the subtree T_1 using postorder, followed by T_2, and so on, until we finish visiting T_n, and then visiting the root r. Note that each subtree is considered as a separate tree, so when applying postorder to a subtree, we apply the procedure for postorder traversal to each of *its* subtrees. [The mantra here is "each root last, traverse subtrees from left to right"]

   Example: Applying postorder to the tree shown above gives the following:

Representing Computations Using Binary Trees: Notice that many mathematical operations (arithmetic, most logical connectives, most set operations) are binary operations (they combine two inputs into a single output). For this reason, we can represent computations using binary trees as follows: The leaves of the tree are the elements being combined. Higher levels in the tree are then labeled with the binary operation being used to combine pairs of elements.

Example: Consider the following computation: $[x + (y \cdot z)] \div [w \cdot (s + t)]$

The following tree represents this computation:



Once we have a tree representing a computation, we can find a expression representing the computation based on the tree in three ways:

1. The **prefix form** [or "Polish" form] for a computation tree is found by traversing the tree using preorder.

   The prefix form for the computation tree shown above is: $\div + x \cdot yz \cdot w + st$

2. The **infix form** [or "Parenthesized" form] for a computation tree is found by traversing the tree using inorder.

   The infix form for the computation tree shown above is: $x + y \cdot z \div w \cdot s + t$

3. The **postfix form** [or "Reverse Polish" form] for a computation is found by traversing the tree using postorder.

   The postfix form for the computation tree shown above is: $xyz \cdot + wst + \cdot \div$