7. Tree Traversal

7.1. Ordered Trees.

DEFINITION 7.1.1. *An* **ordered tree** *is a rooted tree in which the set of children of each vertex is assigned a total order. In a drawing of an ordered rooted tree, with the root shown at the top, the children of a vertex are shown from left to right.*

Discussion

The concept of an ordered tree in some way generalizes the idea of a binary tree in which children of a vertex have been designated as left or right, if we think of the left child of a vertex as being "less than" the right child. The analogy only breaks down for binary trees that are not complete, however, since some vertex may have only a left child, whereas another may only have a right child.

7.2. Universal Address System.

DEFINITION 7.2.1. *The set of vertices of an ordered tree T can be provided with a total order, called a* **universal address system**, *using the following recursive process.*

Basis: *Label the root 0, and label the n children of the root $1, 2, ..., n$ from left to right.*

Recursion: *Given a vertex v with label L at level $k \geq 1$, label its children*

$$L.1, L.2, ..., L.n_v$$

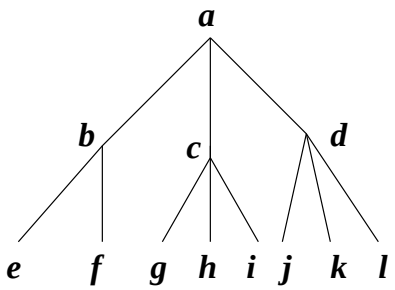
from left to right.

Totally order the vertices of T using the lexicographic ordering.

Discussion

EXAMPLE 7.2.1. *The following table gives the universal address system and the resulting ordering of the vertices for the ordered tree shown below.*

a	$<$	b	$<$	e	$<$	f	$<$	c	$<$	g		
0	$<$	1	$<$	1.1	$<$	1.2	$<$	2	$<$	2.1		
g	$<$	h	$<$	i	$<$	d	$<$	j	$<$	k	$<$	ℓ
2.1	$<$	2.2	$<$	2.3	$<$	3	$<$	3.1	$<$	3.2	$<$	3.3



7.3. Tree Traversal. Suppose we have information stored in an ordered rooted tree. How do we recover information from the tree? That is, how do we visit the vertices in the tree? We shall look at several procedures for visiting, or listing, the vertices of the tree. Each procedure is defined recursively on the subtrees, and each is based on a path that proceeds to the leftmost child of a vertex that has not occurred in the path before moving to a vertex to the right. These algorithms only differ as to *when* the root of a subtree is visited (or listed) relative to the vertices of its subtrees.

DEFINITION 7.3.1. *A procedure used to systematically visit each vertex in a tree is called a* **traversal algorithm**, *or a* **traversal**.

Notice that a subtree of T that does not include the root must have fewer vertices. Therefore, the recursive definition makes sense. Just keep applying the recursion step until you get to the leaves. Once you are down to a subtree that consists only of a leaf apply the basis step.

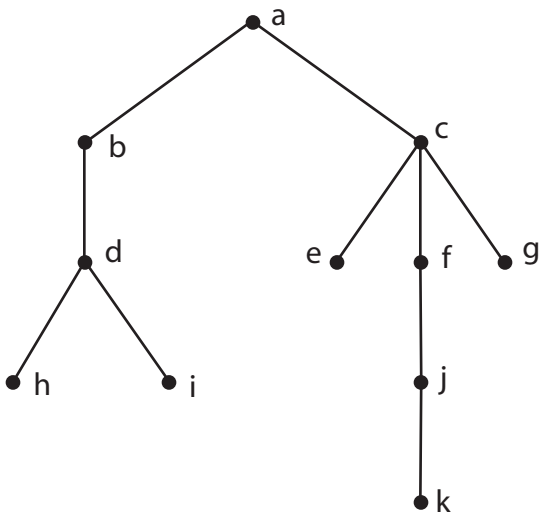
7.4. Preorder Traversal.

DEFINITION 7.4.1. *The* **preorder traversal** *of a rooted tree, T, with n vertices is defined recursively as follows:*

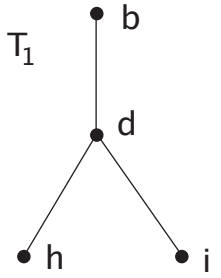
Basis: *If $n = 1$, then the root is the only vertex, so we visit the root.*
Recursive Step: *When $n > 1$ consider the subtrees, $T_1, T_2, T_3, \ldots, T_k$ of T whose roots are all the children of the root of T. Visit each of these subtrees from left to right.*

Discussion

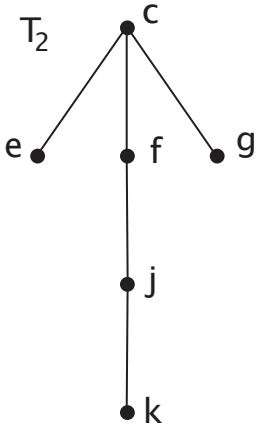
EXAMPLE 7.4.1. *In the Preorder Traversal of the the vertices in the following tree the vertices are visited in the following order: a, b, d, h, i, c, e, f, j, k, g.*



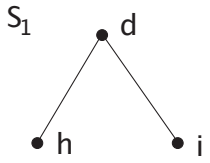
To begin finding the preorder traversal we begin with a. Next find the preorder traversal for the subtree



List that traversal and then find and list the prerorder traversal for the subtree



Now to find the preorder traversal of the subtree, T_1 we start with b and find the preorder traversal of



The preorder traversal of this one is *d, h, i.* The basis step was finally used to find the preorder traversal of S_1's subtrees.

The preorder traversal of T_2 still needs to be found. It is *c, e, f, j, k, g.* By putting all the vertices together in the order in which they were listed we get the preorder traversal *a, b, d, h, i, c, e, f, j, k, g.*

One recommendation is to take the original graph and point to each vertex in the order listed in the preorder traversal to help coordinate the order they are given in the list to their place in the tree.

Notice that if a tree is ordered using the universal address system, then a listing of the vertices in "increasing" order is a preorder traversal of the tree.

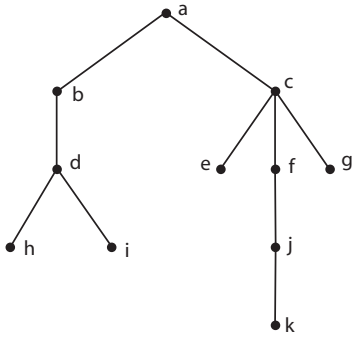
7.5. Inorder Traversal.

DEFINITION 7.5.1. *In an* **inorder traversal** *of a tree the root of a tree or subtree is visited after the vertices of the leftmost subtree, but before the vertices of all other subtrees.*

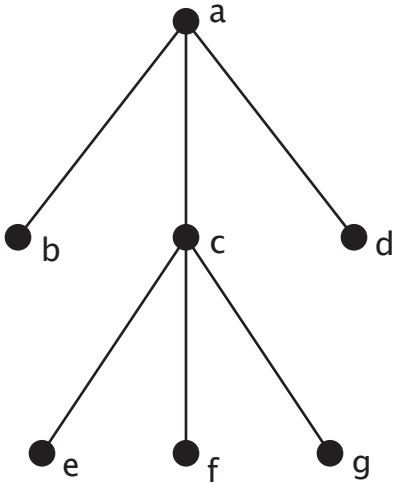
Note that in the inorder traversal, if a vertex, v, has multiple subtrees originating from it v is placed between the vertices of the leftmost subtree and the vertices of all the other subtrees.

EXERCISE 7.5.1. *Give a careful, recursive definition of an inorder traversal.*

EXAMPLE 7.5.1. *The inorder traversal of this tree below is h, d, i, b, a, e, c, k, j, f, g.*



EXAMPLE 7.5.2. *Here is another example to look at. Again, point to each vertex in the order it is listed to develop an understanding of the relationship between inorder traversal and the graph.*

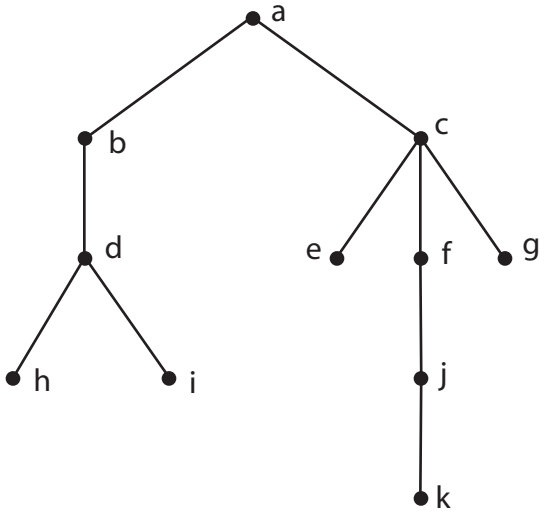


The inorder traversal is b, a, e, c, f, g, d.

7.6. Postorder Traversal.

DEFINITION 7.6.1. *In a* **postorder traversal** *of a tree, the root of a tree/subtree is visited after all of the vertices of its subtrees are visited.*

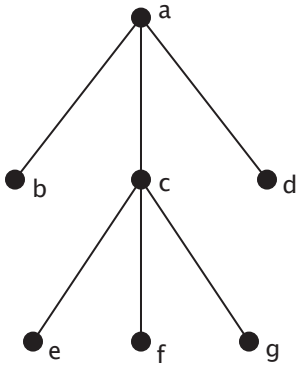
EXAMPLE 7.6.1. *The postorder traversal of the tree below is h, i, d, b, e, k, j, f, g, c, a.*



Discussion

The *postorder traversal* is when the root of a subtree is visited after all its vertices have been visited.

EXAMPLE 7.6.2. *We can find the postorder traversal of the tree*



Postorder Traversal: b, e, f, g, c, d, a.

EXERCISE 7.6.1. *Give a careful, recursive definition of an postorder traversal.*

7.7. Infix Form.

DEFINITIONS 7.7.1.

(1) A fully parenthesized expression is in **infix form**.
(2) The expression obtained by traversing the ordered rooted tree by prefix traversal is **prefix form** *or* **Polish notation**.
(3) The expression obtained by traversing the ordered rooted tree by postfix traversal is **postfix form** *or* **reverse Polish notation**.

EXAMPLE 7.7.1.

$$3(x-y)+\frac{(x+y)^2}{4}$$

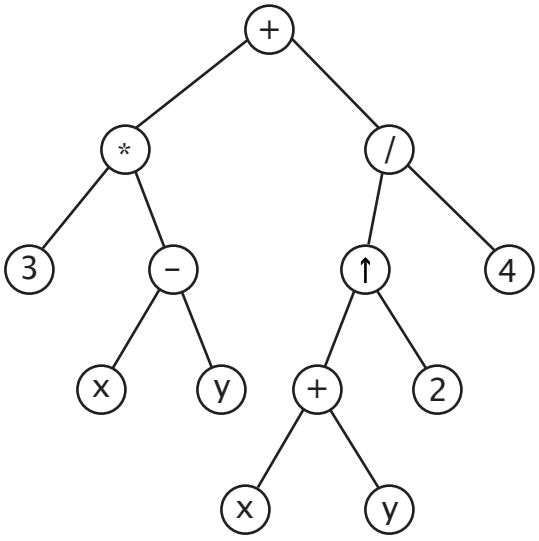
(1) Find the infix form.
(2) Draw the rooted tree for the algebraic expression.
(3) Find the prefix form.
(4) Find the postfix form.

Solutions

(1) The infix form is

$$(3*(x-y))+(((x+y)\uparrow 2)/4)$$

(2) The rooted tree for the algebraic expression is



(3) The prefix form is

$$+ \quad * \quad 3 \quad - \quad x \quad y \quad / \quad \uparrow \quad + \quad x \quad y \quad 2 \quad 4$$

(4) The postfix form is

$$3 \quad x \quad y \quad - \quad * \quad x \quad y \quad + \quad 2 \quad \uparrow \quad 4 \quad / \quad +$$

Discussion

An algebraic representation consists of a finite number of

- variables and numbers
- binary operations: addition $+$, subtraction $-$, multiplication $*$, division $/$, exponentiation $\uparrow$.

Pay attention to the fact that the symbol $\uparrow$ is used for exponentiation in this material. For example, $2^3 = 2 \uparrow 3$.

In these notes we define the infix, prefix, and postfix forms of an algebraic expression and give an example. The tree referred to in example 7.7.1 is found as follows.

A binary ordered tree can be built having internal vertices labeled by the operators and leaves labeled by the variables and numbers. We start from the innermost expressions and work our way outward constructing subtrees from the innermost expressions. These are joined together to form larger subtrees using the operations that join the innermost expressions. The *inorder* listing of the vertices then reproduces the

expression provided the parentheses are inserted as follows: as you begin to traverse a subtree from an internal vertex (operation) insert an open parenthesis; as you leave the subtree insert a closed parenthesis.

Prefix and postfix notation can be found from the tree. Notice that the prefix and postfix notations do *not* require parenthesis. A algebraic expression in prefix or postfix notation is unambiguous. Infix form requires parentheses, however, in order to resolve ambiguities. Despite this fact, infix notation is pretty much the universally accepted form for writing algebraic expressions. We have adopted a convention, so well known that we take it for granted, that allows us to reduce, but not eliminate, the number of parentheses needed without creating ambiguities. Namely, we agree that, in the absence of parentheses, the binary operations are performed in the following order: exponentiation, multiplication, division, addition, subtraction.

EXERCISE 7.7.1. *Find the prefix and postfix forms for the algebraic expressions* $((a * b) + (c/(d \uparrow 3)))$ *and* $(a * (((b + c)/d) \uparrow 3))$.

EXAMPLE 7.7.2. *Find the infix form of the expression given in prefix form.*

$$- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad / \quad * \quad 2 \quad + \quad x \quad y \quad 3$$

Solution

(1) $- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad / \quad * \quad 2 \quad (+ \quad x \quad y) \quad 3$
(2) $- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad / \quad * \quad 2 \quad (x + y) \quad 3$
(3) $- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad / \quad (* \quad 2 \quad (x + y)) \quad 3$
(4) $- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad / \quad (2 * (x + y)) \quad 3$
(5) $- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad (/ \quad (2 * (x + y)) \quad 3)$
(6) $- \quad \uparrow \quad + \quad x \quad y \quad 5 \quad ((2 * (x + y))/3)$
(7) $- \quad \uparrow \quad (+ \quad x \quad y) \quad 5 \quad ((2 * (x + y))/3)$
(8) $- \quad \uparrow \quad (x + y) \quad 5 \quad ((2 * (x + y))/3)$
(9) $- \quad (\uparrow \quad (x + y) \quad 5) \quad ((2 * (x + y))/3)$
(10) $- \quad ((x + y) \uparrow 5) \quad ((2 * (x + y))/3)$
(11) $(- \quad ((x + y) \uparrow 5) \quad ((2 * (x + y))/3))$
(12) $(((x + y) \uparrow 5) - ((2 * (x + y))/3))$

To go from prefix form to infix form, we read the expression right from left. Look for the rightmost operation and the variables or numbers immediately to the right of it. Put parenthesis around these and change to infix notation. Move to the next right most operation and put parenthesis around it and the expressions just to the right of it. Change to infix notation, and so on.

EXAMPLE 7.7.3. *Find the value of the postfix expression*

$$6 \quad 2 \quad \uparrow \quad 9 \quad / \quad 7 \quad 2 \quad 3 \quad \uparrow \quad + \quad 5 \quad / \quad +$$

Solution

(1) $(6 \quad 2 \quad \uparrow) \quad 9 \quad / \quad 7 \quad 2 \quad 3 \quad \uparrow \quad + \quad 5 \quad / \quad +$
(2) $(6^2) \quad 9 \quad / \quad 7 \quad 2 \quad 3 \quad \uparrow \quad + \quad 5 \quad / \quad +$
(3) $36 \quad 9 \quad / \quad 7 \quad 2 \quad 3 \quad \uparrow \quad + \quad 5 \quad / \quad +$
(4) $(36 \quad 9 \quad /) \quad 7 \quad 2 \quad 3 \quad \uparrow \quad + \quad 5 \quad / \quad +$
(5) $(36/9) \quad 7 \quad 2 \quad 3 \quad \uparrow \quad + \quad 5 \quad / \quad +$
(6) $4 \quad 7 \quad (2 \quad 3 \quad \uparrow) \quad + \quad 5 \quad / \quad +$
(7) $4 \quad 7 \quad 8 \quad + \quad 5 \quad / \quad +$
(8) $4 \quad (7 \quad 8 \quad +) \quad 5 \quad / \quad +$
(9) $4 \quad 15 \quad 5 \quad / \quad +$
(10) $4 \quad (15 \quad 5 \quad /) \quad +$
(11) $4 \quad 3 \quad +$
(12) 7

This time we look for the left most operation and the numbers immediately to the left of it.

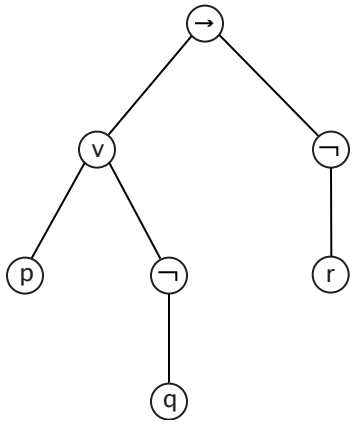
The exact same procedures may be applied to logical expressions as well as set operation expressions.

EXAMPLE 7.7.4. *Given the logical expression* $(p \vee \neg q) \rightarrow \neg r$

1. *Find the infix notation.*
2. *Represent using an ordered rooted tree.*
3. *Find the preorder form.*
4. *Find the postorder form.*

Solution:

1. $(p \vee (\neg q)) \rightarrow (\neg r)$
2. *Tree:*



3. *Preorder:* $\rightarrow \vee p \neg q \neg r$
4. *Postorder:* $pq\neg \vee r\neg \rightarrow$