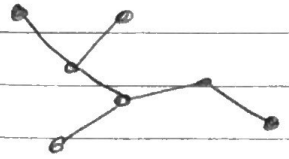
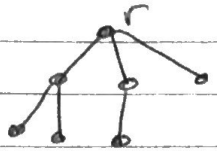
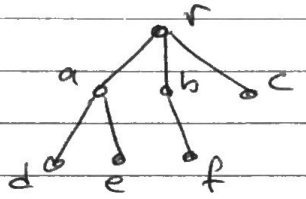
11.3 Tree traversal



a tree (Connected graph with no simple circuits)



a rooted tree (one vertex labeled as root r)



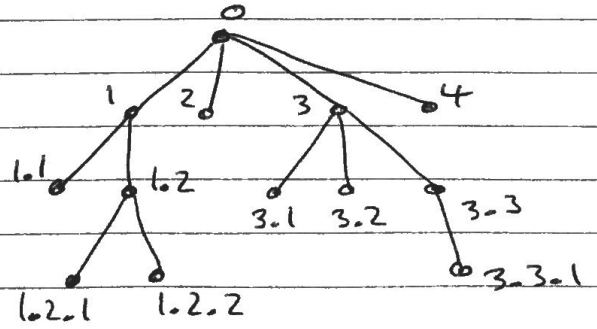
an ordered rooted tree (children labeled and put in order $a < b < c$, $d < e$, shown left to right).

Ordered rooted trees can be complicated structures. We will look at 3 ways to visit every vertex - this is called a tree traversal (imagine an ant again).

Universal address system

In this system the root is labeled as 0 and its children 1, 2, ..., k. Working down the tree, the children of a parent vertex with label L are L.1, L.2, ..., L.m.

For example



Then we can list the vertices in the lexicographic (dictionary) order. Remember this means for words that they come in alphabetical order and, if they start with the same letters, look for the first place they differ

cat < dog, can < cat, can < canary.

In our example

$0 < 1 < 1.1 < 1.2 < 1.2.1 < 1.2.2 < 2 < 3 < \cdots$

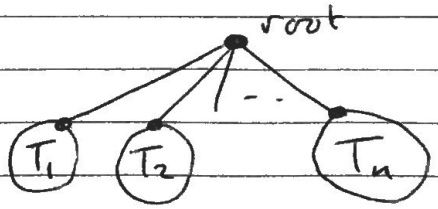
So we can visit each vertex (traverse the tree) in this order.

Example. Suppose a vertex has label 2.6.3.5
Give its level and the label of its parent.

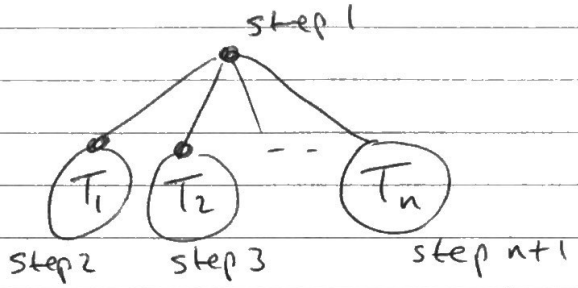
Solution. It must be level 4 as there are four numbers. The parent has label 2.6.3

Preorder We can first describe this method of tree traversal recursively.

Suppose the root has n children. There will be a subtree for each of these children $T_1, T_2, \ldots, T_n$ (with each child its root)

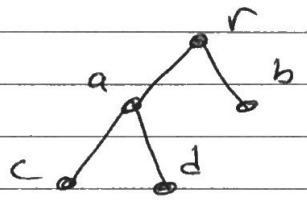


For preorder, visit the root first in step 1. Then visit T_1 in preorder, then T_2 in preorder and so on until T_n in step n+1.



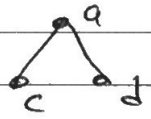
preorder

For example, let T be the ordered rooted tree



T

To traverse T in preorder we visit the root r first. The subtrees are

$T_1 =$  $T_2 =$ • b

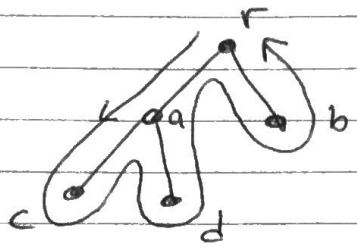
Next traverse T_1 in preorder, starting with its root a. Its subtrees are •c, •d so we do c next then d. Finally we do T_2 in preorder – just the root b.

Altogether, the preorder for T is [r, a, c, d, b]

In fact the universal address system gives the same ordering as preorder.

This recursive definition of preorder is useful for programming. A nice way to understand it uses a curve like

this, starting to the left of the root and following closely around the tree

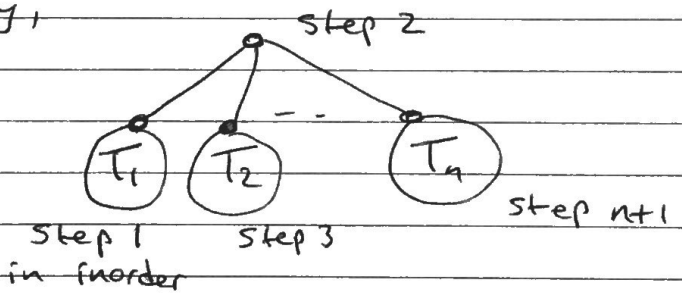


T

If you list a vertex the first time the curve passes it you get the preorder.

Inorder

This method gives a different ordering. Recursively,



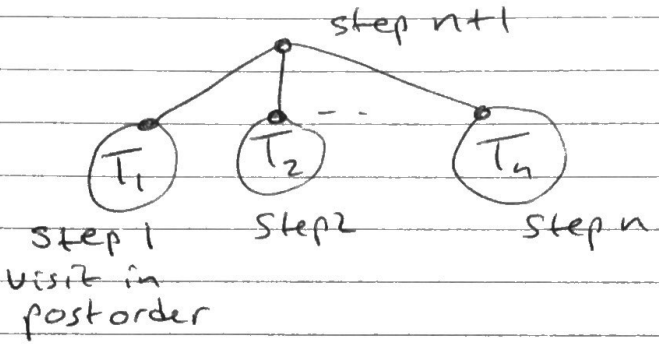
we do the subtree T_1 first in inorder, then the root and then the rest of the subtrees.

Using the curve, we list a leaf the first time it is passed and we list an internal vertex the second time it is passed.

Example. The inorder for T using either description is $\boxed{c, a, d, r, b}$

Postorder

Our final method, recursively

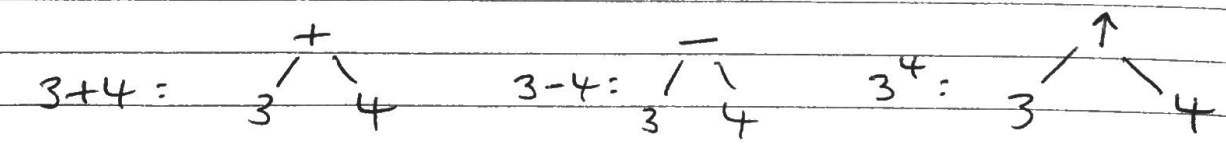


visiting the root last. Using the curve description we list a vertex the last time the curve passes it.

Example The post order for T is c, d, a, b, r

Application to arithmetic, algebra

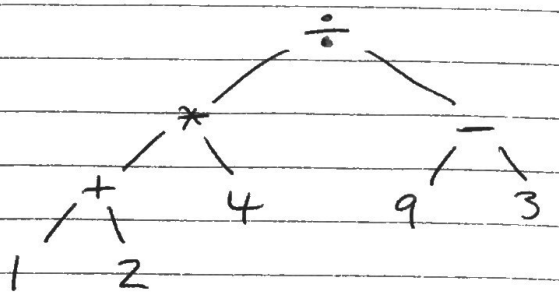
The binary operations of arithmetic and algebra can be represented by simple ordered rooted trees



(using "↑" for exponentiation).

More complicated

$((1+2)*4) \div (9-3)$:



So each internal vertex is an operation and each leaf a number.

(preorder)

Polish notation means the prefix order of this tree: $\div * + 1\ 2\ 4 - 9\ 3$.

We get the original order of the expression (without parentheses) if we use the infix notation, which means the inorder

$$1 + 2 * 4 \div 9 - 3.$$

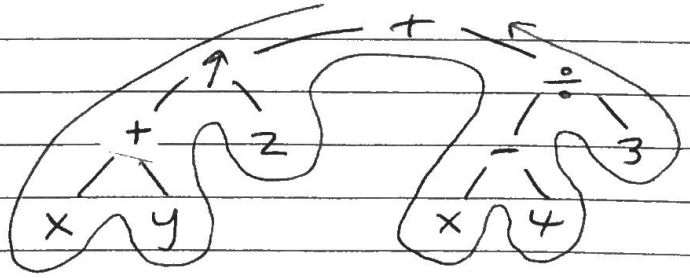
Reverse Polish notation means the postorder of the tree. Also called postfix

$$1\ 2 +\ 4 *\ 9\ 3 -\ \div$$

Example Convert this expression to prefix notation

$$((x+y) \uparrow 2) + ((x-4) \div 3).$$

Solution. First we make the tree



Then following the curve around it gives the preorder

$$+ \uparrow + x\ y\ 2 \div - x\ 4\ 3.$$

Note that prefix notation gives a way to express algebra without parentheses. An operation is always applied to the two following parts

$+ \; x \; 8$ means $x+8$ in prefix.

So finding these and working back shows

$$+ \; - \; * \; 2 \; - \; 9 \; 6 \; 4 \; 3$$

means

$$+ \; - \; * \; 2 \; \underbrace{- \; 9 \; 6}_{3} \; 4 \; 3$$

$$\underbrace{\quad\quad\quad}_{6}$$

$$\underbrace{\quad\quad\quad\quad\quad}_{2}$$

$$5$$

So the result is 5. Similarly, postfix notation has the operation following the two parts

$x \; 8 \; +$ means $x+8$ in postfix.

Example Evaluate this postfix expression

$$7 \; 2 \; 3 \; * \; - \; 4 \; \uparrow \; 9 \; 3 \; \div \; +$$

Try it. You should get 4.