Tree Traversal
Discrete Mathematics

Why Tree Traversal

Ordered rooted trees are often used to store information. We need procedures for visiting each vertex of an ordered rooted tree to access data.

In ordered rooted trees, the children of an internal vertex are shown from left to right in the drawings representing these directed graphs

Finding a method to visit all the vertices of an ordered rooted tree is equivalent to build a compatible total order of the vertices.

Universal Address System

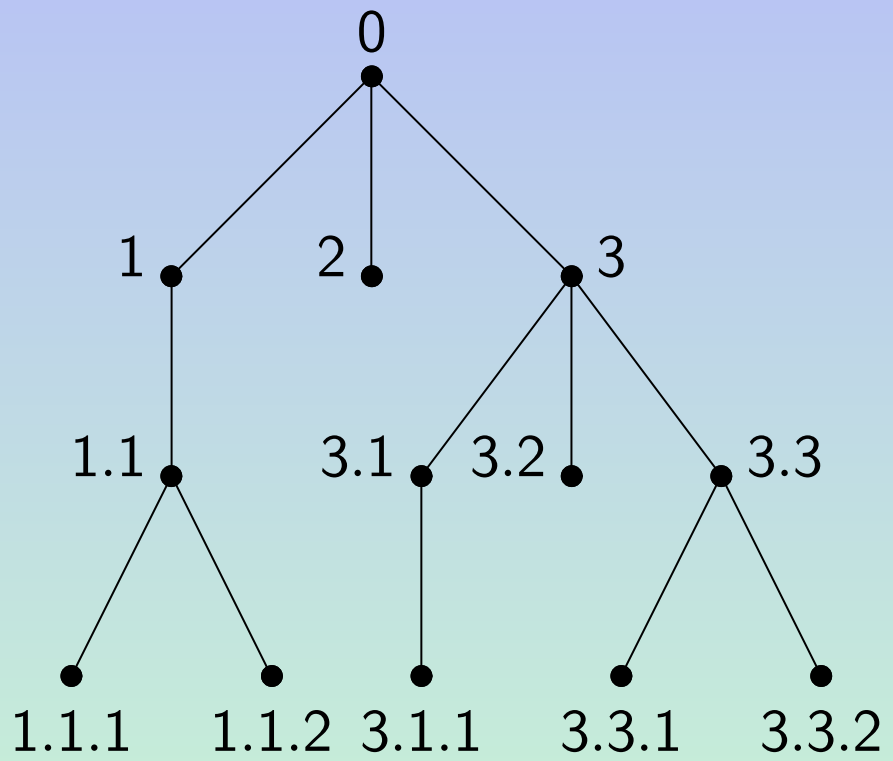
We label recursively the vertices of the ordered rooted tree as follows:

1. Label the root with the integer 0. Then label its k children (at level 1) form left to right with $1, 2, 3, ..., k$.

2. For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with $A.1, A.2, ..., A.k_v$.

Following this procedure, a vertex v at level n, for $n \geq 1$, is labeled $x_1.x_2.\cdots.x_n$, where the unique path from the root to v goes through the x_1st vertex at level 1, the x_2nd vertex at level 2, and so on. This labeling is called the **universal address system** of the ordered rooted tree.

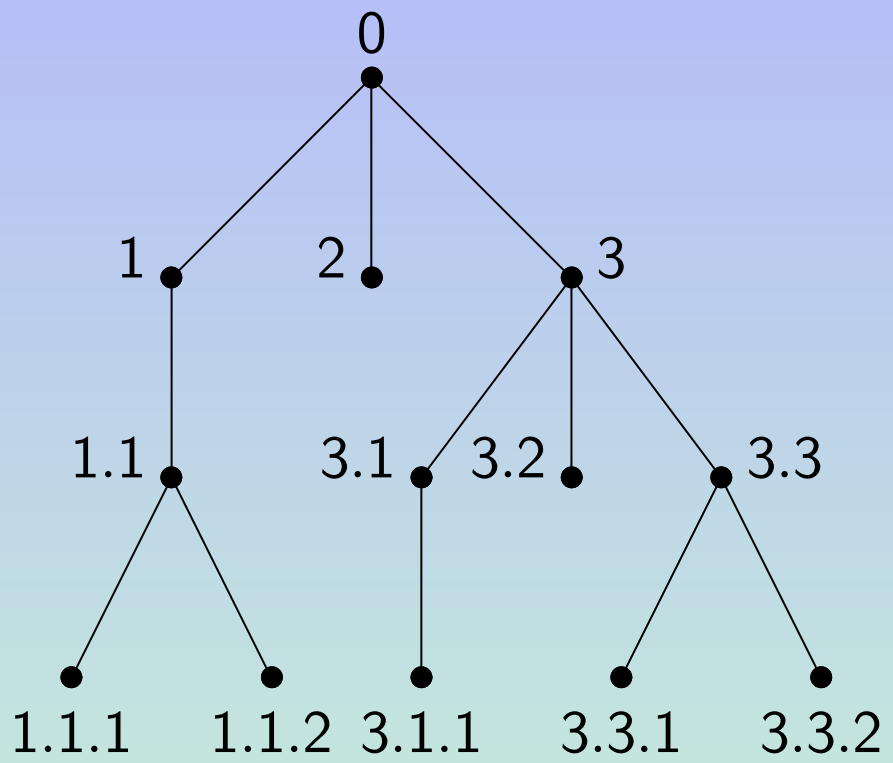
Example of the Universal Address System



Lexicographic Ordering

We can totally order the vertices using the **lexicographic ordering** of their labels in the universal address system. The vertex labeled $x_1.x_2.\cdots.x_n$ is less than the vertex labeled $y_1.y_2.\cdots.y_m$ if there is an i, $0 \leq i \leq n$ with $x_1 = y_1$, $x_2 = y_2$, ..., $x_{i-1} = y_{i-1}$ and $x_i < y_i$; or if $n < m$ and $x_i = y_i$ for $i = 1, 2, ..., n$.

Example of the Universal Address System



Lexicographic ordering of the ordered rooted tree vertices:
$0 < 1 < 1.1 < 1.1.1 < 1.1.2 < 2 < 3 < 3.1 < 3.1.1 < 3.2 < 3.3 < 3.3.1 < 3.3.2$

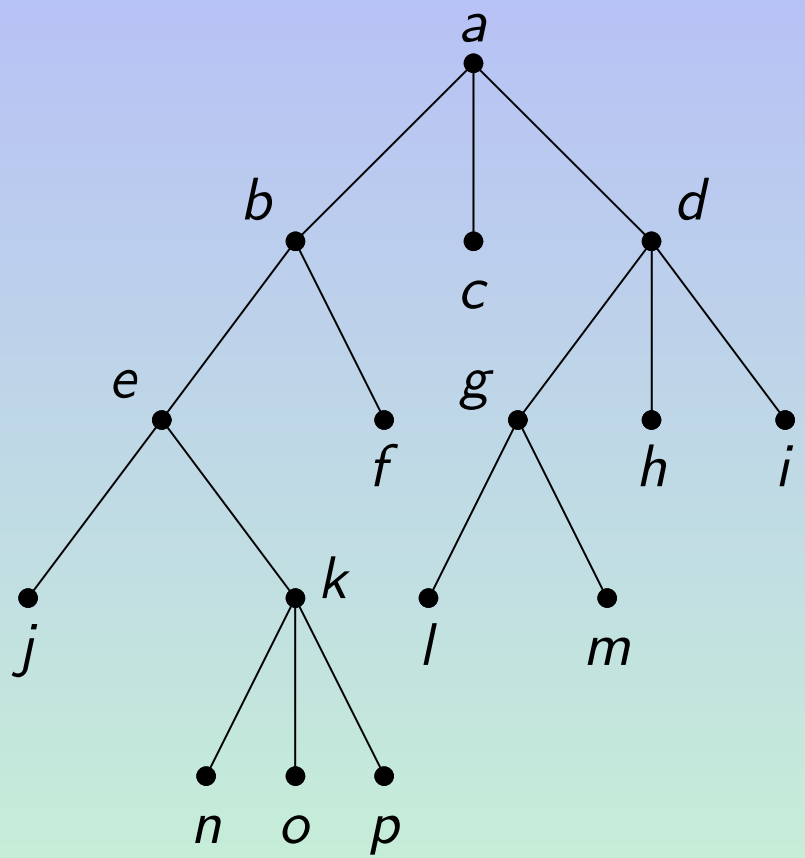
Preorder Traversal

Definition

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **preorder traversal** of T. Otherwise, suppose that T_1, T_2, ..., T_n are the subtrees at r from left to right in T. The **preorder traversal** begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.

The preorder traversal of an ordered rooted tree gives the same ordering of the vertices as the ordering obtained using a universal address system.

Example of Preorder Traversal



Preorder traversal: *a b e j k n o p f c d g l m h i* (Depth-first order).

Recursive Algorithm: Preorder Traversal

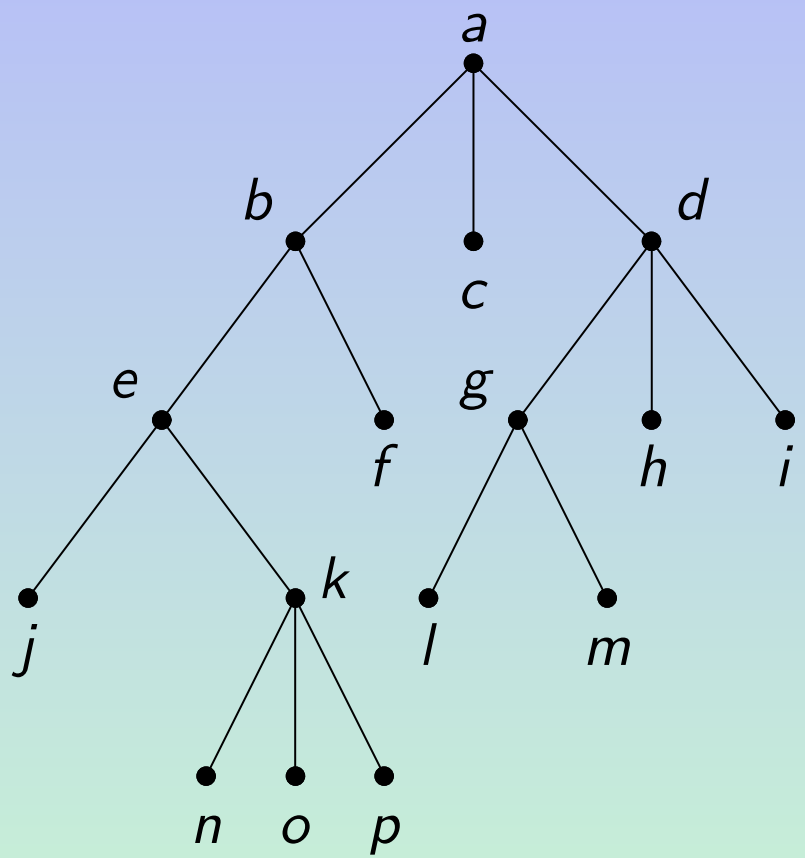
procedure *preorder* (T: ordered rooted tree)
$r :=$ root of T
Add r to the *list*
for each child c of r from left to right
begin
    $T(c) :=$ subtree with c as its root
    preorder$(T(c))$
end
{*list* contains the preorder traversal of T}

Inorder Traversal

Definition

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **inorder traversal** of T. Otherwise, suppose that T_1, T_2, ..., T_n are the subtrees at r from left to right. The **inorder traversal** begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, then T_3 in inorder, ..., and finally T_n in inorder.

Example of Inorder Traversal



Inorder traversal: *j e n k o p b f a c l g m d h i* (symmetric order)

Recursive Algorithm: Inorder Traversal

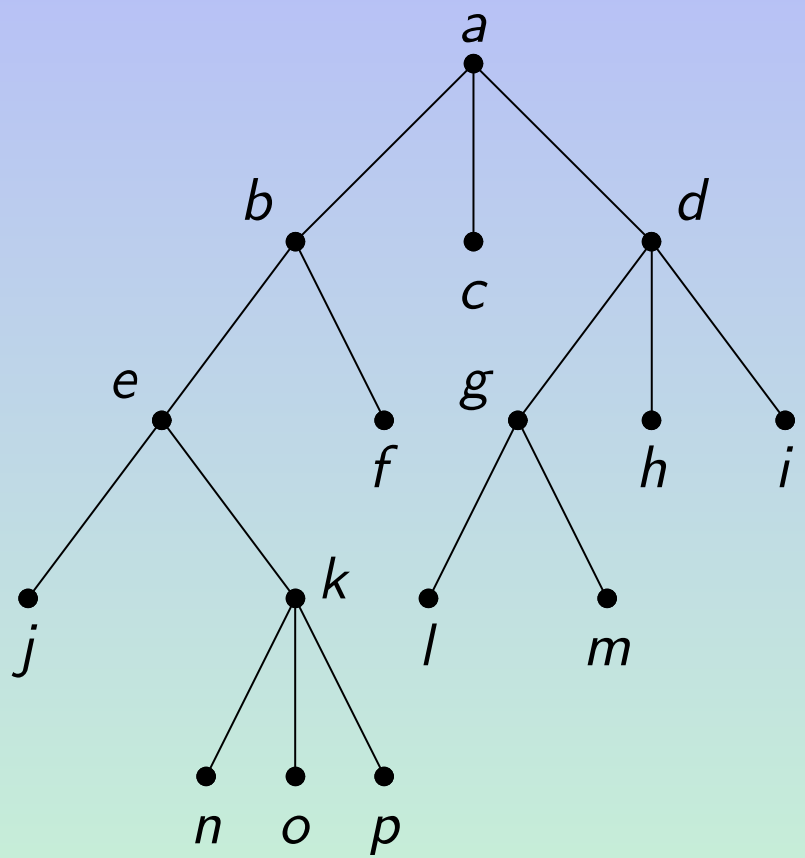
procedure *inorder* (T: ordered rooted tree)
$r :=$ root of T
if r is a leaf **then** add r to the *list*
else
begin
    $l :=$ first child of r from left to right
    $T(l) :=$ subtree with l as its root
    inorder$(T(l))$
    Add r to the *list*
    for each child c of r except for l from left to right
    begin
        $T(c) :=$ subtree with c as its root
        inorder$(T(c))$
    end
end
{*list* contains the inorder traversal of T}

Postorder Traversal

Definition

Let T be ans ordered rooted tree with root r. If T consists only of r, then r is the **postorder traversal** of T. Otherwise, suppose that T_1, T_2, ..., T_n are the subtree at r from left to right. The **postorder traversal** begins by traversing T_1 in postorder, then T_2 in postorder, ..., then T_n in postorder, and ends by visiting r.

Example of Postorder Traversal



Postorder traversal: $j\ n\ o\ p\ k\ e\ f\ b\ c\ l\ m\ g\ h\ i\ d\ a$.

Recursive Algorithm: Postorder Traversal

procedure *postorder* (T: ordered rooted tree)
$r :=$ root of T
for each child c of r from left to right
begin
    $T(c) :=$ subtree with c as its root
    postorder$(T(c))$
end
Add r to the *list*
{*list* contains the postorder traversal of T}

Preorder, Inorder and Postorder Traversal

To traverse a non-empty tree in **preorder**, perform the following operations recursively at each node, starting with the root node:

1. Visit the root.
2. Traverse all the subtrees form left to right.

To traverse a non-empty tree in **inorder**, perform the following operations recursively at each node:

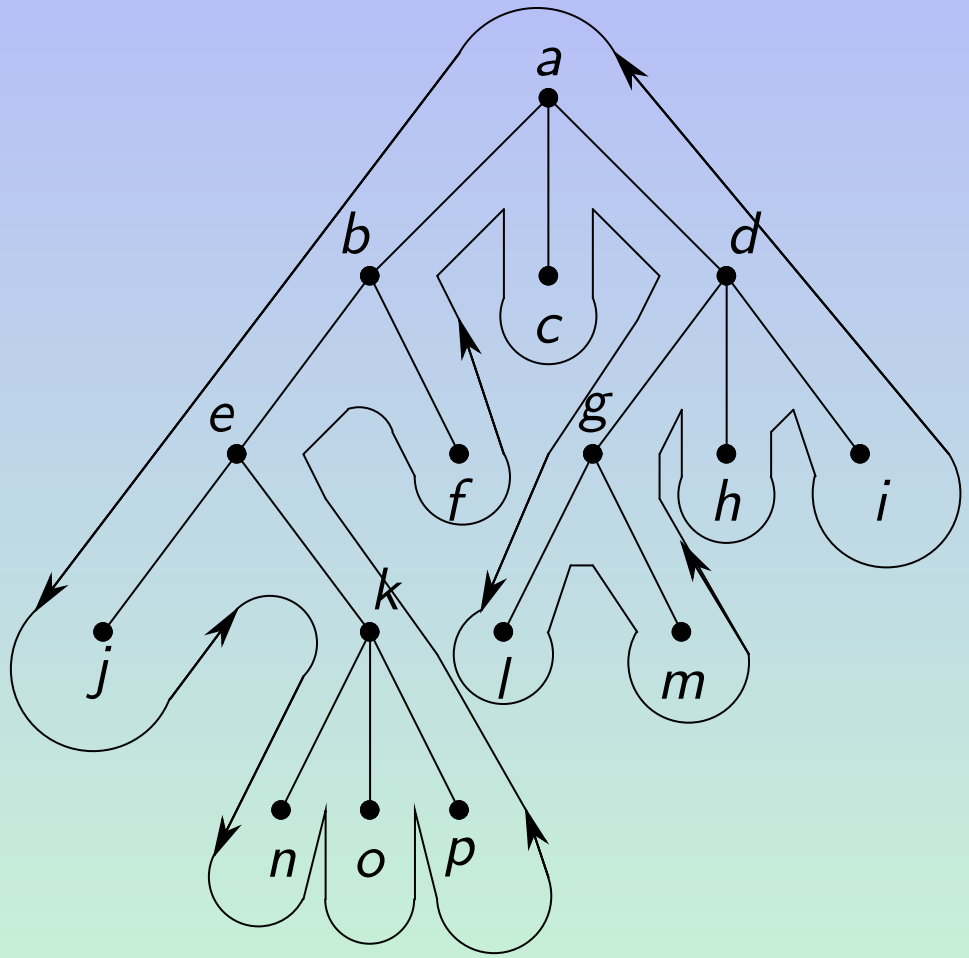
1. Traverse the leftmost subtree.
2. Visit the root.
3. Traverse all the remaining subtrees from left to right.

To traverse a non-empty tree in **postorder**, perform the following operations recursively at each node:

1. Traverse all the subtrees from left to right
2. Visit the root.

Preorder, Inorder and Postorder Traversal

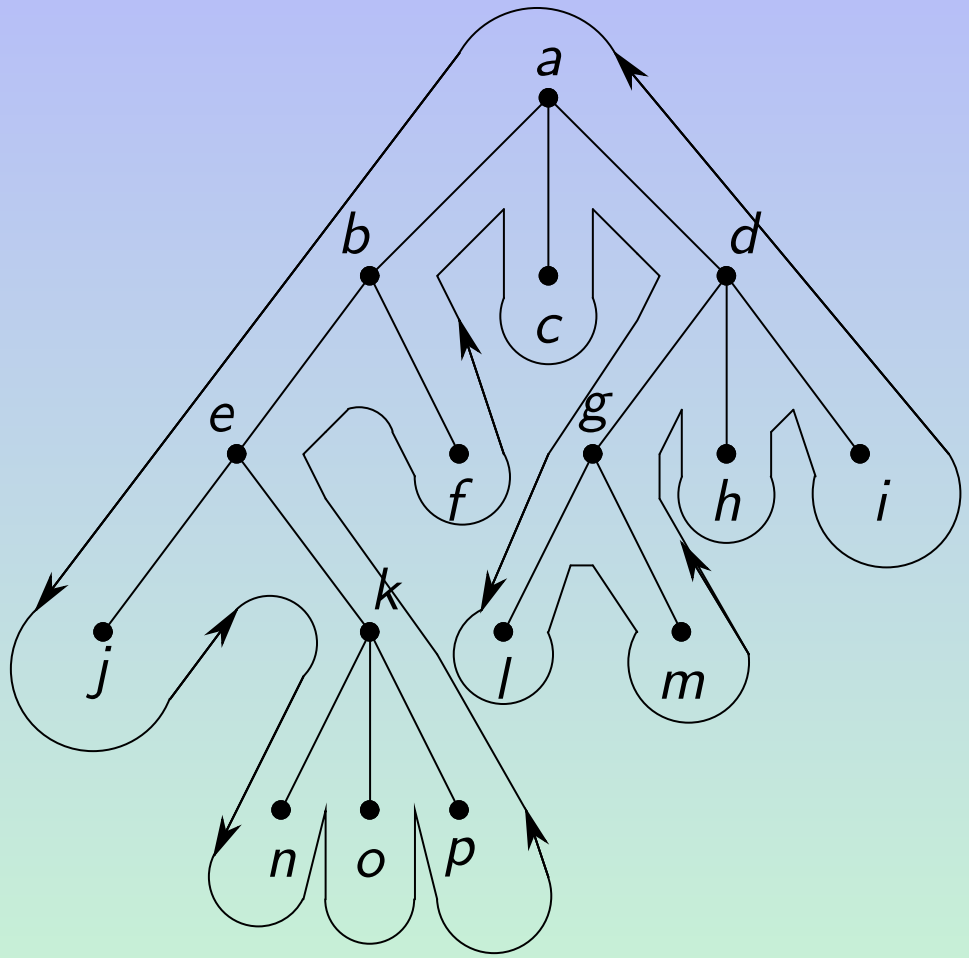
Draw a curve around the ordered rooted tree starting at the root, moving along the edges, in counterclockwise direction.



Preorder Traversal

We can list the vertices in **preorder** by listing each vertex the first time this curve passes it.

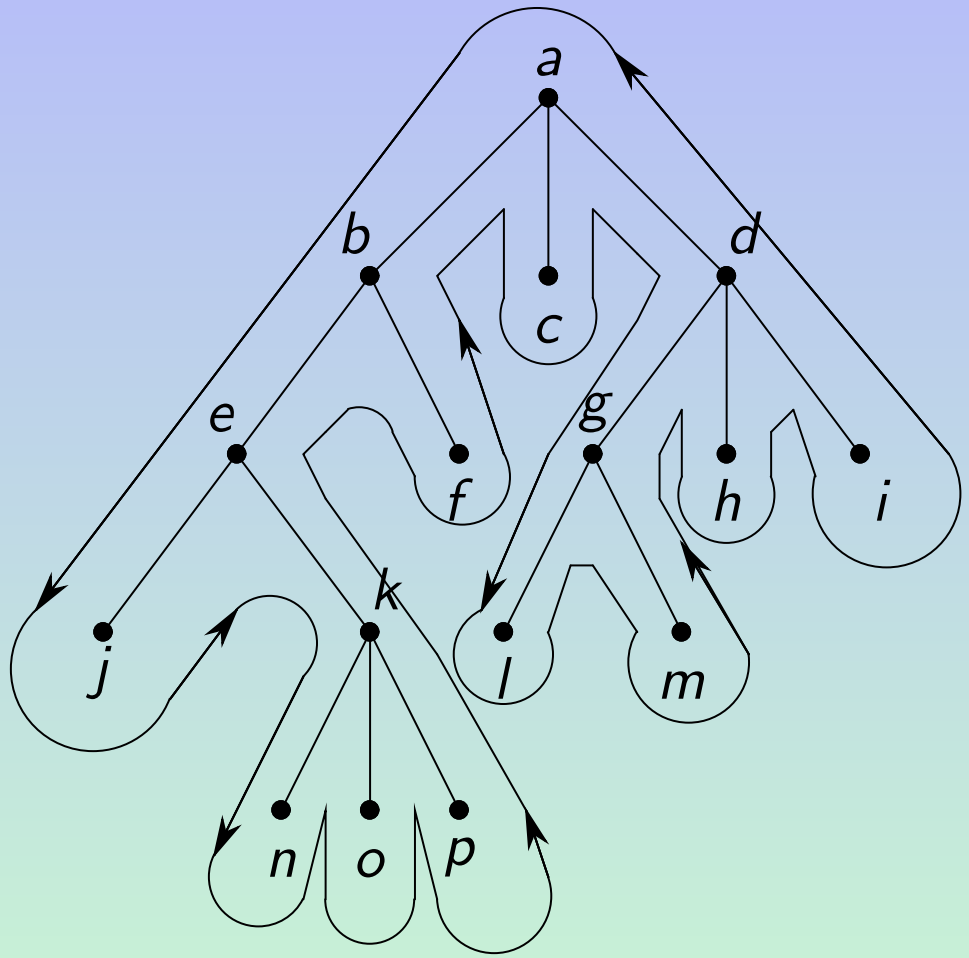
a b e j k n o p f c d g l m h i



Inorder Traversal

We can list the vertices in **inorder** by listing a leaf the first time the curve passes it and listing each internal vertex the second time the curve passes it.

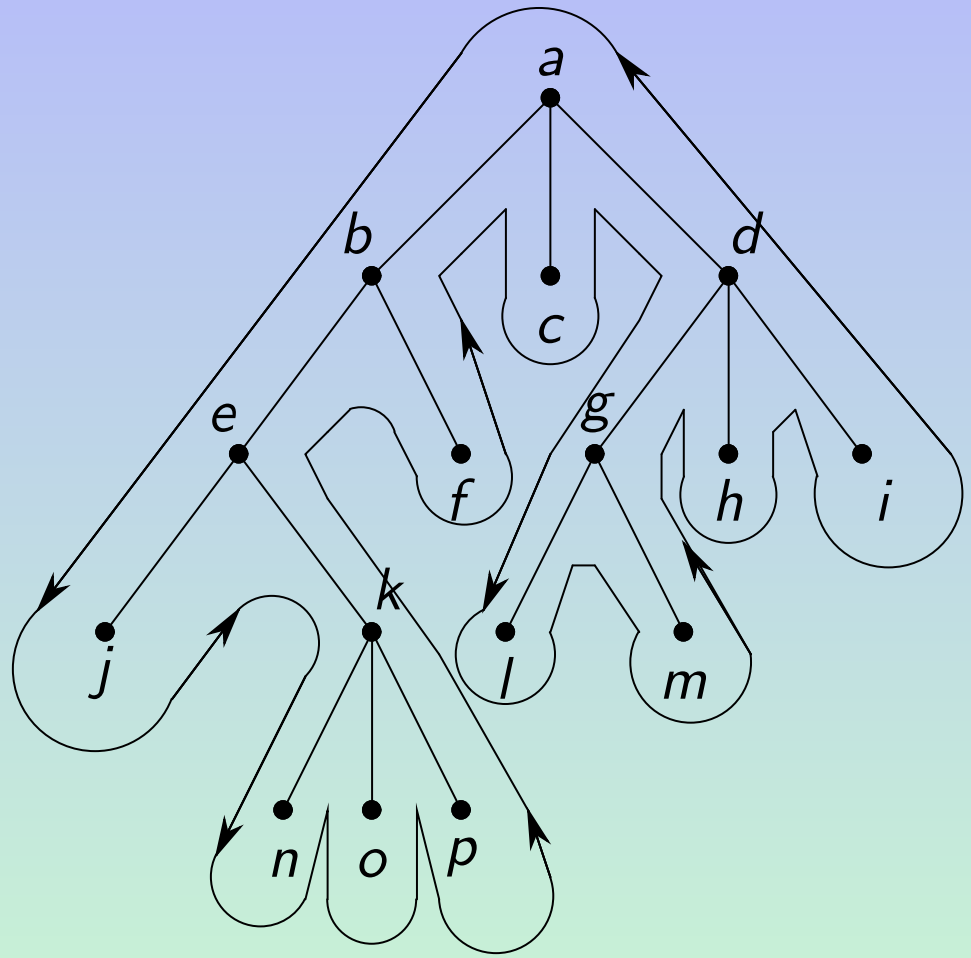
j e n k o p b f a c l g m d h i



Postorder Traversal

We can list the vertices in **postorder** by listing a vertex the last time it is passed on the way back to its parent.

j n o p k e f b c l m g h i d a



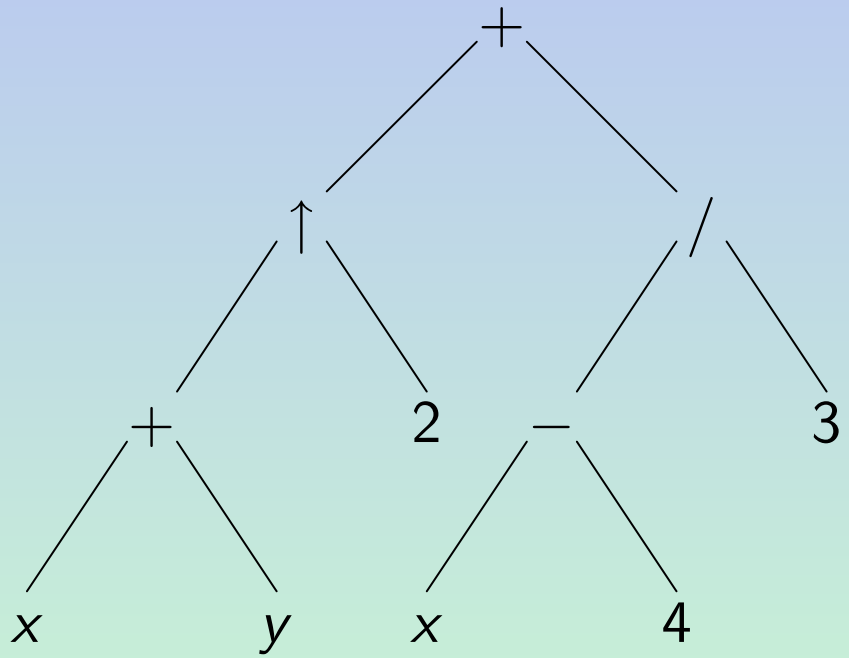
Example: Infix, Prefix and Postfix Notation for Arithmetic Expressions

An arithmetic expression with the operators $+$, $-$, $*$, $/$ and $\uparrow$ (for exponentiation) can be represented using an ordered rooted tree.

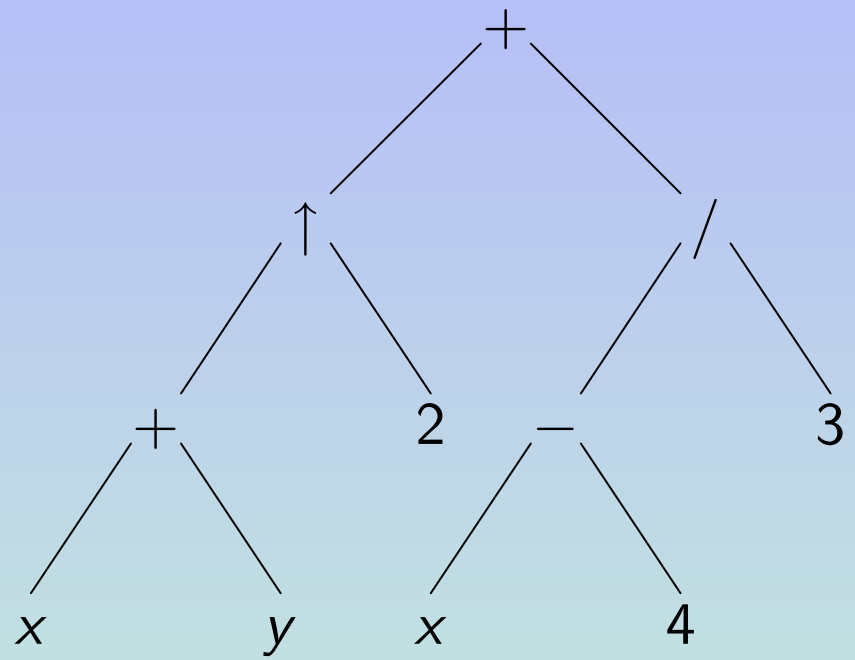
In an ordered rooted tree, the internal vertices represent operations and the leaves represent the variables or numbers. Each operation operates on its left and right subtrees, in that order.

Binary Tree of an Expression

A binary tree representing $((x + y) \uparrow 2) + ((x - 4)/3)$:



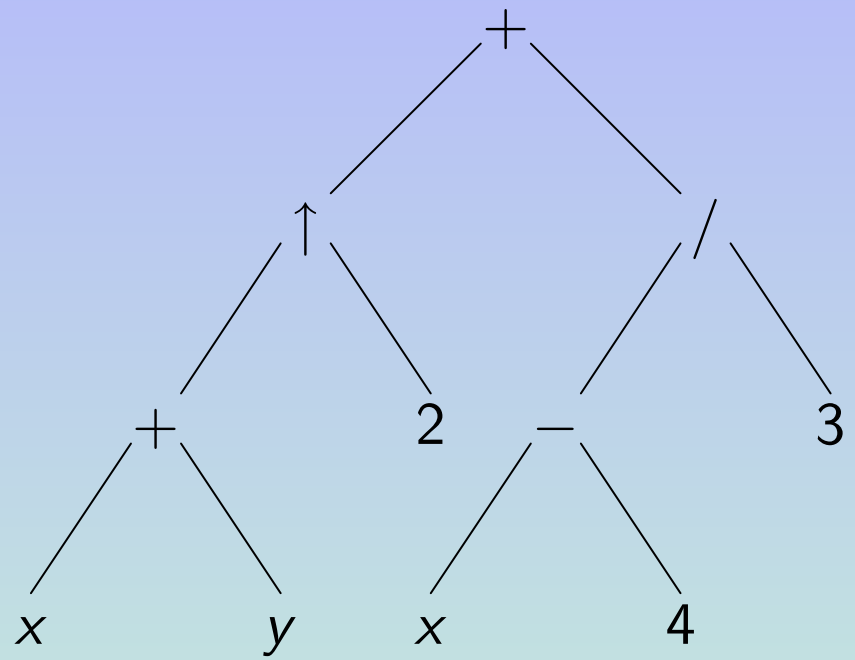
Infix Form of an Expression



Inorder traversal: $x + y \uparrow 2 + x - 4 / 3$

To make such expression unambiguous, it is necessary to include parentheses in the inorder traversal whenever we encounter an operation. The fully parenthesized expression obtained in this way is the **infix form**: $(((x + y) \uparrow 2) + ((x - 4)/3)))$

Prefix Form of an Expression



Preorder traversal: $+ \uparrow + x\, y\, 2\, / \, - \, x\, 4\, 3$

The **prefix form** of an expression is obtained by using the preorder traversal of its ordered rooted tree. Expressions written in prefix form are said to be in **Polish notation**, which are unambiguous (and so do not need parentheses).

Historical Note: Jan Lukasiewicz

Born on 21 December 1878 in Lvov.

Died on 13 February 1956 in Dublin, Ireland.

`www-groups.dcs.st-and.ac.uk/~history/Mathematicians/Lukasiewicz.html`

How to Evaluate a Prefix Form

In a prefix form, a binary operator precedes its two operands.

Prefix forms are evaluated from *right to left*.

Whenever an operation is performed, we consider the result as a new operand.

How to Evaluate a Prefix Form

Evaluate the prefix form $+-*235/\uparrow 234$

$$+ \quad - \quad * \quad 2 \quad 3 \quad 5 \quad / \quad \underbrace{\uparrow \quad 2 \quad 3}_{2\uparrow 3=8} \quad 4$$

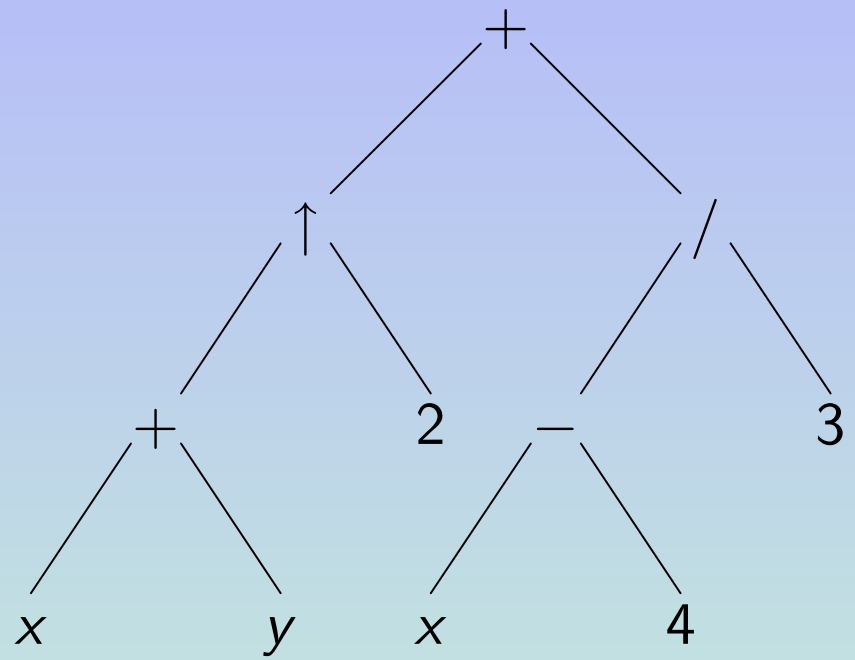
$$+ \quad - \quad * \quad 2 \quad 3 \quad 5 \quad \underbrace{/ \quad 8 \quad 4}_{8/4=2}$$

$$+ \quad - \quad \underbrace{* \quad 2 \quad 3}_{2*3=6} \quad 5 \quad 2$$

$$+ \quad \underbrace{- \quad 6 \quad 5}_{6-5=1} \quad 2$$

$$\underbrace{+ \quad 1 \quad 2}_{1+2=3}$$

Postfix Form of an Expression



Postorder traversal: $x\,y\,+\,2\,\uparrow\,x\,4\,-\,3\,/\,+$

The **postfix form** of an expression is obtained by using the postorder traversal of its ordered rooted tree. Expressions written in postfix form are said to be in **Reverse Polish notation**, which are unambiguous (and so do not need parentheses).

How to Evaluate a Postfix Form

In a postfix form, a binary operator follows its two operands.

Postfix forms are evaluated from *left to right*.

Whenever an operation is performed, we consider the result as a new operand.

How to Evaluate a Postfix Form

Evaluate the postfix form $723*-4\uparrow 93/+$

$$7\ \underbrace{2\ 3\ *}_{2*3=6}\ -\ 4\ \uparrow\ 9\ 3\ /\ +$$

$$\underbrace{7\ 6\ -}_{7-6=1}\ 4\ \uparrow\ 9\ 3\ /\ +$$

$$\underbrace{1\ 4\ \uparrow}_{1\uparrow 4=1}\ 9\ 3\ /\ +$$

$$1\ \underbrace{9\ 3\ /}_{9/3=3}\ +$$

$$\underbrace{1\ 3\ +}_{1+3=4}$$

RPN (Reverse Polish Notation) Calculator

perso.wanadoo.fr/noel.jouenne/calc.html