Lesson 3. Trees.

1. Definitions, Properties, and Examples.
2. Rooted Trees.
3. Tree Traversal.

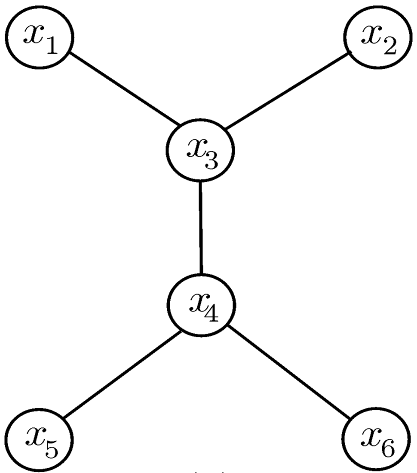
1. Definitions, Properties, and Examples.

Let G = (V, E) be a loop-free undirected graph.

DEFINITIONS:

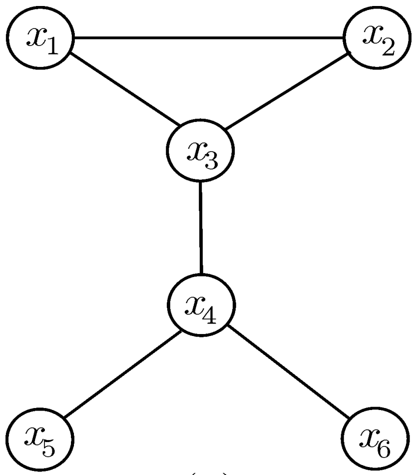
1. The graph G is called a **tree** if G is **connected** and contains **no cycles**. When a graph is a tree we write T instead of G to emphasize this structure.

EXAMPLE:



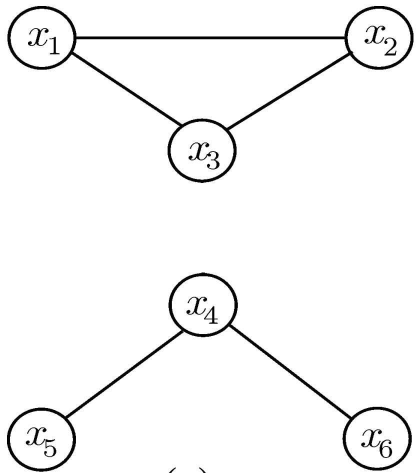
(a)

It's a tree because is connected and contains no cycles.



(b)

It is not a tree because it contains the cycle $\mathbf{x_1x_2x_3x_1}$.



(c)

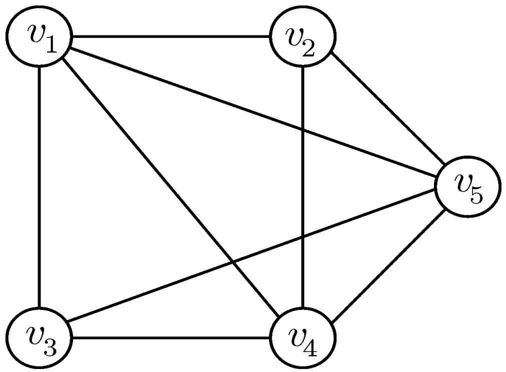
It is not a tree because it is not connected.

1. Definitions, Properties, and Examples.

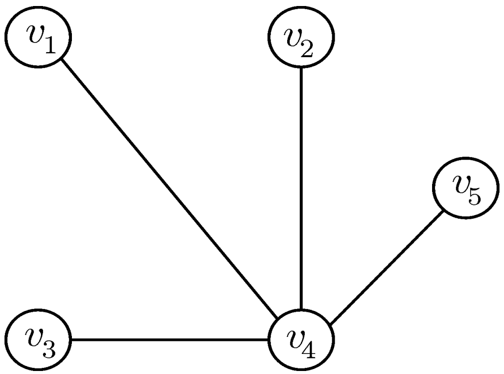
2. A **spanning tree** T for a connected graph G is a spanning subgraph (contains all the vertices of G) that is also a tree.

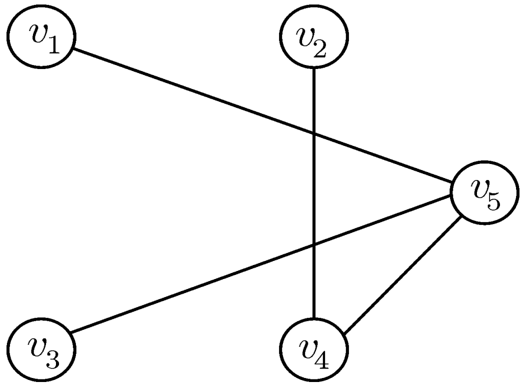
EXAMPLE:

Let's consider



The following trees are spanning trees of G:





1. Definitions, Properties, and Examples.

THEOREM 1

1. If a, b are distinct vertices in a tree T = (V, E), then **there is a unique path** that connects these vertices.
2. If G = (V, E) is an undirected graph, then G is connected if and only if G has a spanning tree.
3. In every tree T = (V, E), **$|V| = |E| + 1$**. ($|V|$ denotes the cardinality of V).
4. For every tree T = (V, E), if $|V| \geq 2$, then T has at least two vertices with degree 1.

Example

A tree has one vertex of degree 6, 2 of degree 3, 3 of degree 2, and the rest of degree 1. Compute the total number of vertices in the tree. The graph does not have any other type of vertices except those indicated. State the theorems you use to calculate it.

k = number of vertices with degree 1

All the graphs satisfy:

$$\sum_{v \in V} \deg(v) = 2|E|$$

It's a tree, then |E|=|V|-1

$$\sum_{v \in V} \deg(v) = 2(|V| -1) \longrightarrow 1\cdot 6 + 2\cdot 3 + 3\cdot 2 + k\cdot 1 = 2(6+k-1)$$

$$\sum_{v \in V} \deg(v) = 1\cdot 6 + 2\cdot 3 + 3\cdot 2 + k\cdot 1$$

$$|V| = 1 + 2 + 3 + k = 6+k$$

18+k = 10 +2k

k=8

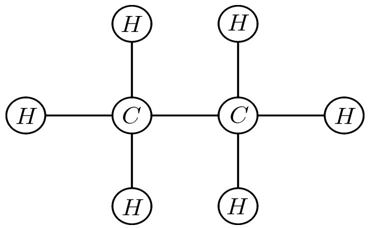
8 vertices with degree 1

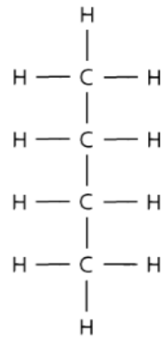
Solution:
8 +1+2+3 = 14 vertices

1. Definitions, Properties, and Examples.

EXAMPLE: Saturated Hydrocarbons and Trees. Graphs can be used to represent molecules, where atoms are represented by vertices and bonds between them by edges. In graph models of saturated hydrocarbons, each **carbon atom** is represented by a **vertex of degree 4**, and each **hydrogen atom** is represented by a **vertex of degree 1**.

- **If a saturated has n carbon atoms, show that it has 2n + 2 hydrogen atoms**.





Considering the saturated hydrocarbon as a tree T = (V, E), let **k** equal the number of pendant vertices, or **hydrogen atoms**, in the tree. Then with a total of **n + k** vertices, where each of the **n** carbon atoms has degree 4, we find that

$$\sum_{v\in V} \deg(v) = 2|E| \longrightarrow \sum_{v\in V} \deg(v) = 2(|V|-1) \longrightarrow 4n+k=2(n+k-1) \longrightarrow 4n+k=2n+2k-2$$

$$\boxed{|V| = |E| + 1 \longrightarrow |E|=|V|-1}$$

$$\boxed{\begin{array}{l}\sum_{v\in V} \deg(v)=4n+k \\ |V|=n+k\end{array}}$$

$$\boxed{2n+2=k}$$

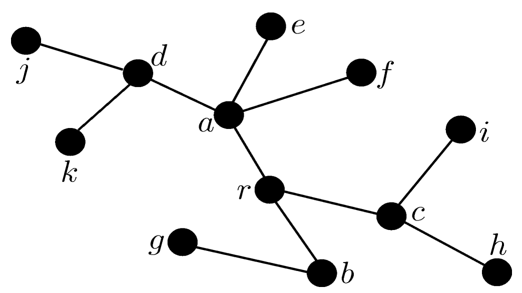
2. Rooted Trees.

DEFINITION: Let T be a tree.

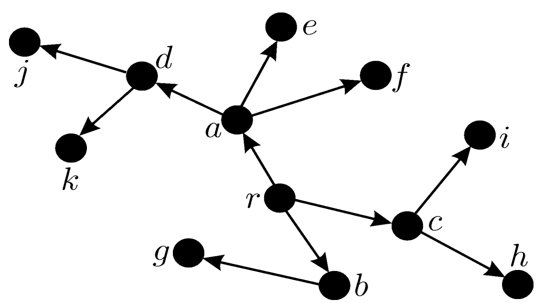
Choosing a vertex r_0 of T, the **rooted tree** with root r_0, denoted by $T(r_0)$ is the directed graph obtained directing each edge **away from the root**.

Remark: We can direct each edge away from the root because there is a unique path from the root to each vertex of the graph (by Theorem 1).

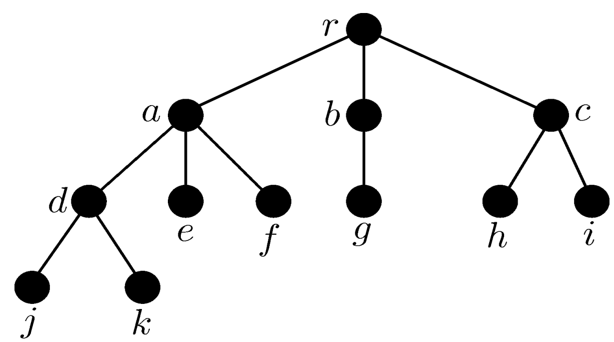
EXAMPLE: Consider the tree T. Choosing the root r.



T



T(r)

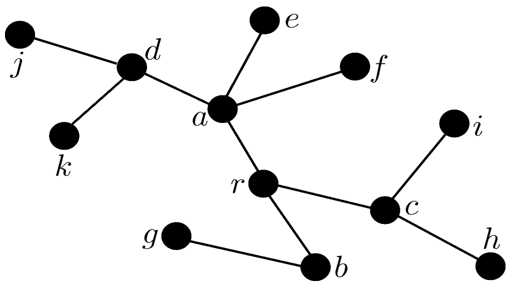


T(r)

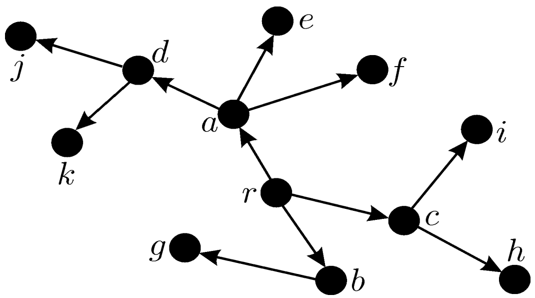
2. Rooted Trees.

Remarks:

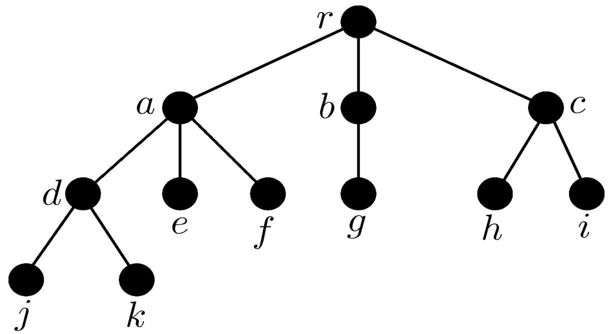
- We usually draw a rooted tree with its root at the top of the graph.
- The **arrows** indicating the directions of the edges in a rooted tree can be **omitted**, because the choice of root determines the directions of the edges.



T

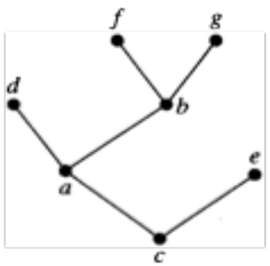


T(r_0)

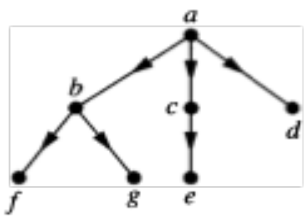


T(r_0)

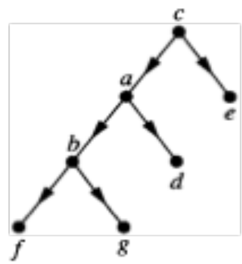
- Note that different choices of the root produce **different** rooted trees.



T



With root a: T(a)



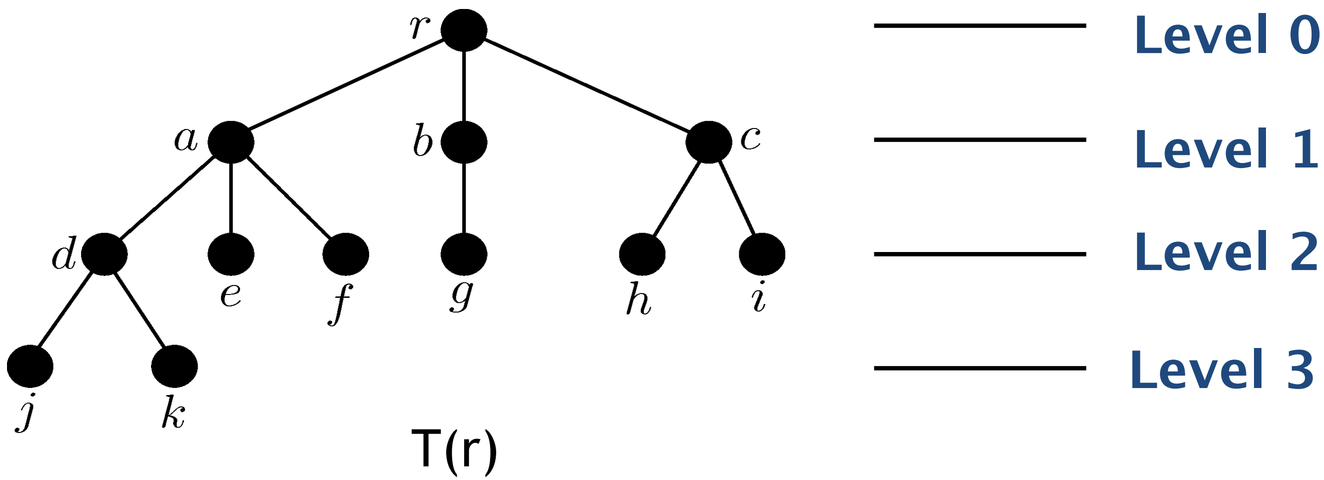
With root c: T(c)

2. Rooted Trees.

DEFINITION: Let **T** be a rooted tree and let **u** be a vertex of **T**.

- The **level of u** is the length of the path form the root to **u**. The level of the root is defined to be zero.
- The **height of a rooted tree** is the maximum of the levels of vertices. In other words, the height of a rooted tree is the length of the longest path from the root to any vertex.

EXAMPLE:



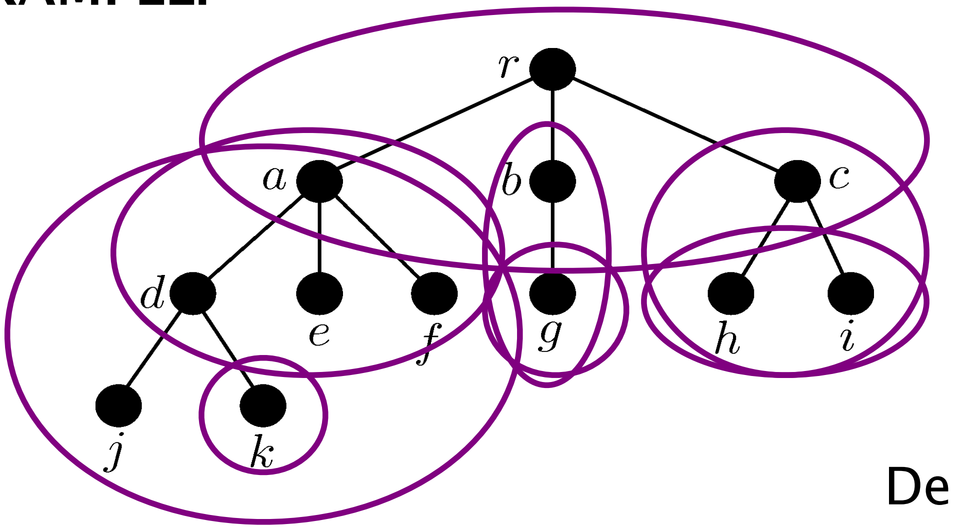
Height (maximum level): 3

2. Rooted Trees.

DEFINITION: Let **T** be a rooted tree with root $\mathbf{r_0}$. Let x, y, z be vertices of T, and let $\mathbf{v_0 v_1 \ldots v_{n-1} v_n}$ be a path in **T**. Then:

- $\mathbf{v_{n-1}}$ is called the **parent** of $\mathbf{v_n}$ (the parent of a vertex is unique).
- $\mathbf{v_n}$ is called a **child** of $\mathbf{v_{n-1}}$.
- $\mathbf{v_0 v_1 \ldots v_{n-1}}$ are are called the **ancestors** of $\mathbf{v_n}$.
- If **x** is an ancestor of **y**, then **y** is called a **descendant** of **x**.
- Vertices with the same parent are called **siblings**.

EXAMPLE:



a is the parent of d, e, f

c is the parent of h, i

The ancestors of k are d, a, r

g is a child of b

a, b, c are children of r

h, i are siblings

g has not siblings

Descendants of a: d, e, f, j, k

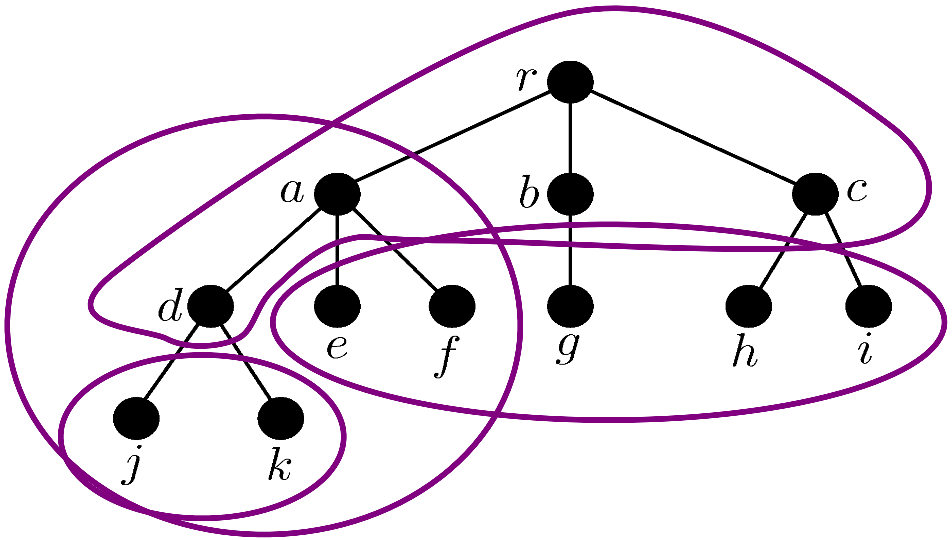
Descendants of r: all the vertices

2. Rooted Trees.

DEFINITION: Let **T** be a rooted tree with root $\mathbf{r_0}$. Then:

- A vertex with outdegree 0 (i.e. if it has not children) is called a **leaf** (or **terminal vertex**).
- All other vertices are called **branch** nodes (or **internal** vertices).
- If **v** is any vertex of the tree, the **subtree** at **v** is the subgraph induced by the root v and all of its descendants (there may be none).

EXAMPLE:



Terminal vertices: j, k, e, f, g, h, i

Internal vertices : *d*, *a*, *b*, *c*, *r*

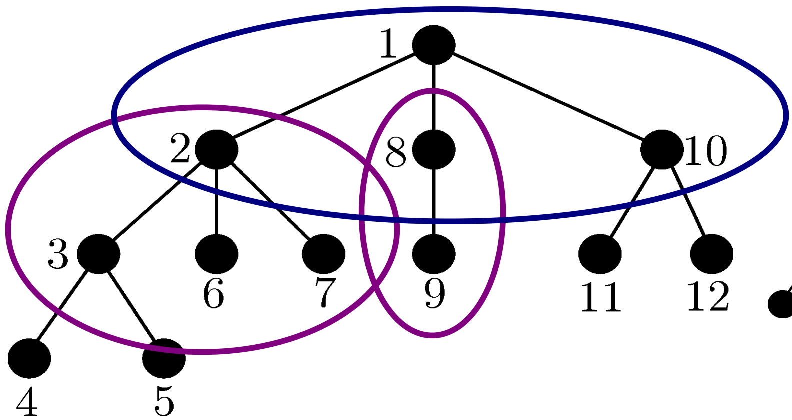
Subtree at *a*

2. Rooted Trees.

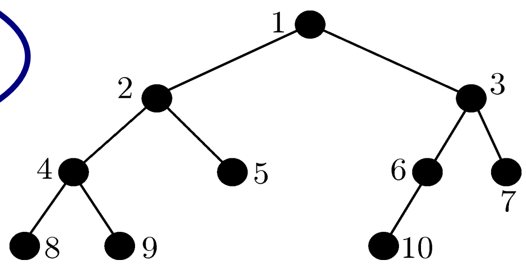
DEFINITIONS:

1. An **ordered rooted tree** is a rooted tree where **the children** of each internal vertex **are ordered**. Ordered rooted trees are drawn so that the children of each internal vertex are shown in **order from left to right**.
2. An ordered **binary** tree (usually called just a **binary tree**), is an ordered rooted tree where each vertex has **at most two children**.
3. If each internal vertex of a binary tree has **exactly two children** (the left child and the right child), then the rooted tree is called a **complete binary tree**.

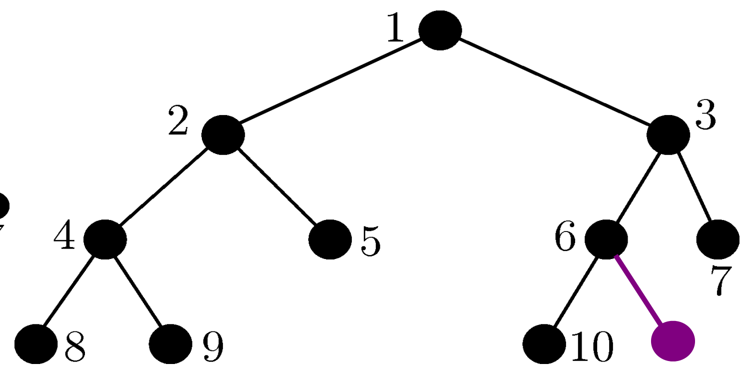
EXAMPLE:



Non-binary rooted tree



Non-complete binary tree



Complete binary tree

2. Rooted Trees.

THEOREM:

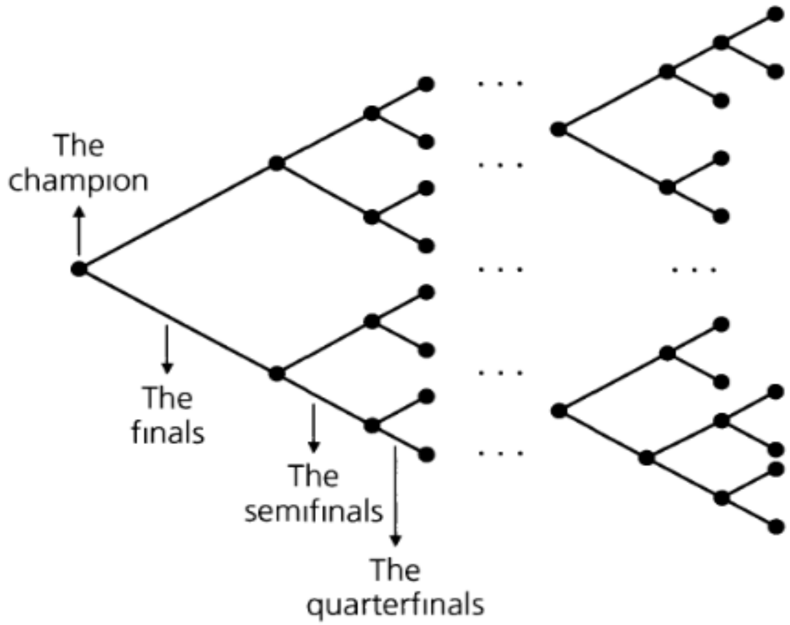
Let T be a complete binary tree with i internal vertices, then $|V|=2i+1$ and T has $i+1$ terminal vertices.

THEOREM:

Let T be a binary tree with height h and t terminal vertices, then $t \leq 2^h$.

EXAMPLE:

The Wimbledon tennis championship is a single-elimination tournament wherein a player (or doubles team) is eliminated after a single loss. If 27 women compete in the singles championship, how many matches must be played to determine the number-one female player?



- Consider the tree shown in Figure. With **27 women competing**, there are **27 leaves** in this complete binary tree.
- **Number of women = Number of terminal vertices=27.**
- **Number of matches = Number of internal vertices.**
- So from above Theorem the number of internal vertices (which is the number of matches) is i, with **27= i+1**. **Then i=26**.

3. Tree Traversal.

Introduction

Ordered rooted trees are often used to **store information**. We need procedures for visiting each vertex of an ordered rooted tree to access data. We will describe several important algorithms for visiting all the vertices of an ordered rooted tree.

Ordered rooted trees can also be used to represent various types of expressions, such as arithmetic expressions involving numbers, variables, and operations. The different listings of the vertices of ordered rooted trees used to represent expressions are useful in the evaluation of these expressions.

Traversal Algorithms

Procedures for systematically visiting every vertex of an ordered rooted tree are called traversal algorithms. We will describe three of the most commonly used such algorithms, **preorder traversal**, **inorder traversal**, and **postorder traversal**. Each of these algorithms can be defined recursively.

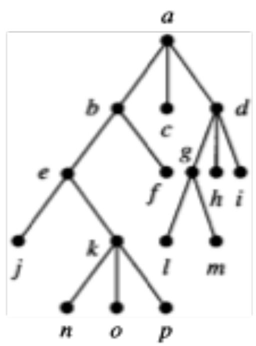
3. Tree Traversal.

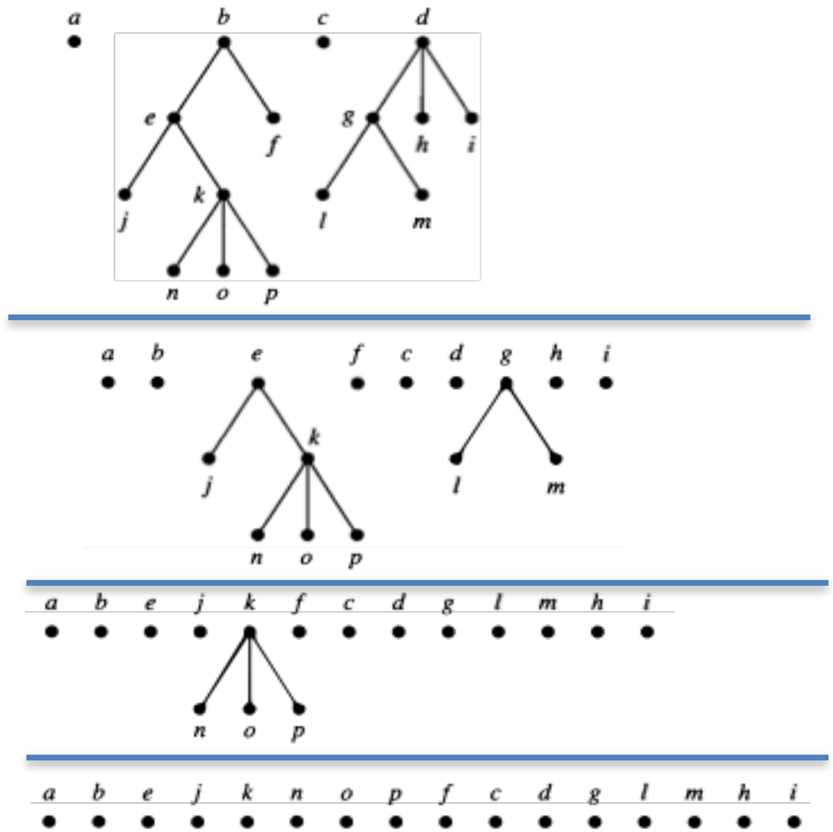
DEFINITION

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 $T_1, T_2, \ldots, T_n$ 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.

<u>EXAMPLE:</u> Preorder traversal:

Visit root, visit subtrees left to right





3. Tree Traversal.

ALGORITHM Preorder Traversal.

procedure preorder(T : ordered rooted tree) r := root of T
list r
for each child c of r from left to right
begin
    T(c) := subtree with c as its root
    preorder(T(c))
end

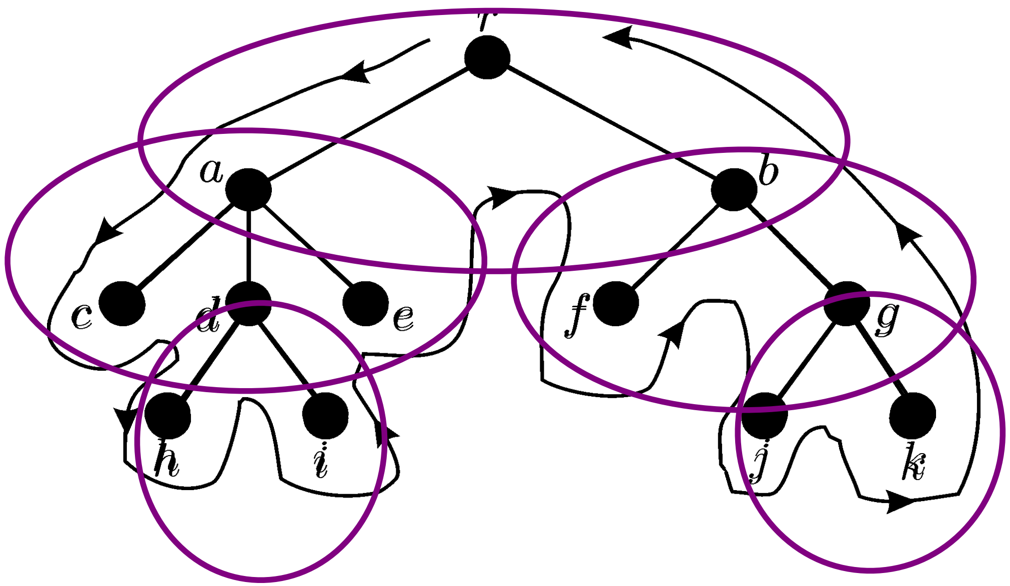
EXAMPLE:

$T_r \equiv r\ T_a\ T_b$

$\equiv r\ a\ T_c\ T_d\ T_e\ b\ T_f\ T_g$

$\equiv r\ a\ c\ d\ T_h\ T_i\ e\ b\ f\ g\ T_j\ T_k$

$\equiv r\ a\ c\ d\ h\ i\ e\ b\ f\ g\ j\ k$

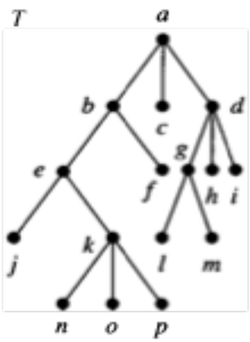


3. Tree Traversal.

DEFINITION

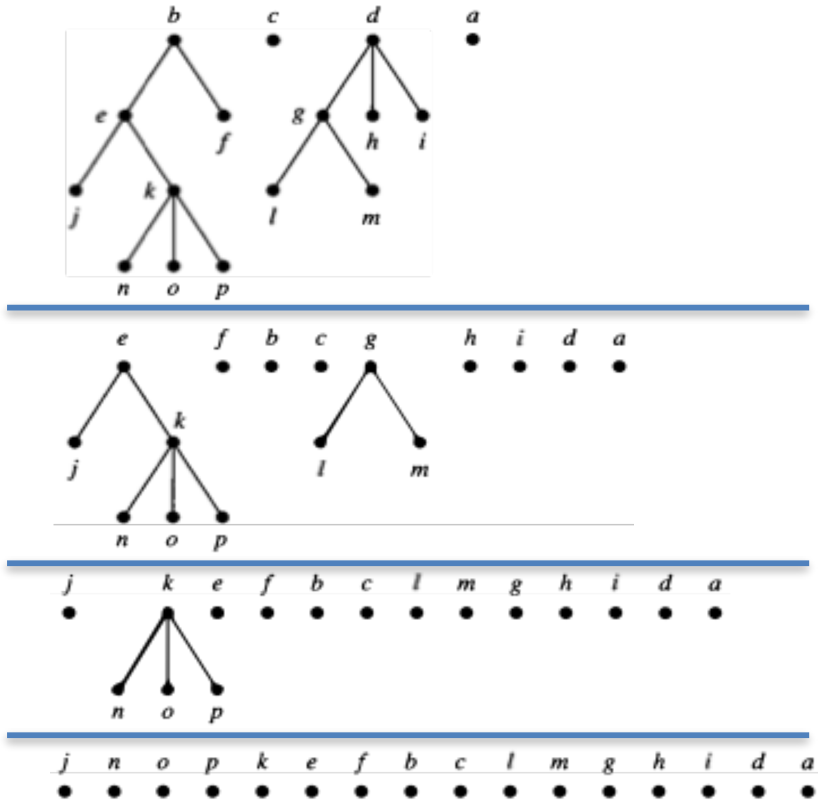
Let T be an ordered rooted tree with root r. If T consists only of r, then r is the postorder traversal of T. Otherwise, suppose that T_1 , T_2 , . . . , T_n are the subtrees at r from left to right. The **postorder traversal** begins by traversing T_1 in postorder, then T_2 in postorder, . . . , then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

EXEMPLE:



Postorder traversal :

Visit subtrees left to right; visit root



3. Tree Traversal.

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

list r

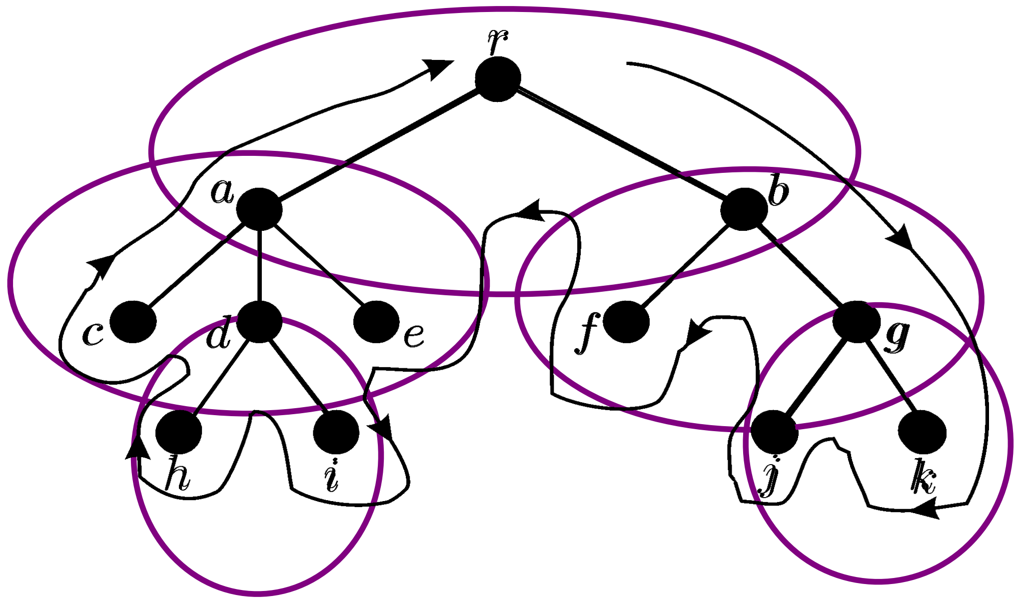
EXAMPLE:

$T_r \equiv \underline{T_a\ T_b}\ r$

$\equiv \underline{T_c\ T_d\ T_e}\ a\ \underline{T_f\ T_g}\ b\ r$

$\equiv c\ \underline{T_h\ T_i}\ d\ e\ a\ f\ \underline{T_j\ T_k}\ g\ b\ r$

$\equiv c\ h\ i\ d\ e\ a\ f\ j\ k\ g\ b\ r$

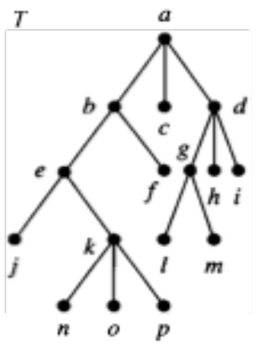


3. Tree Traversal.

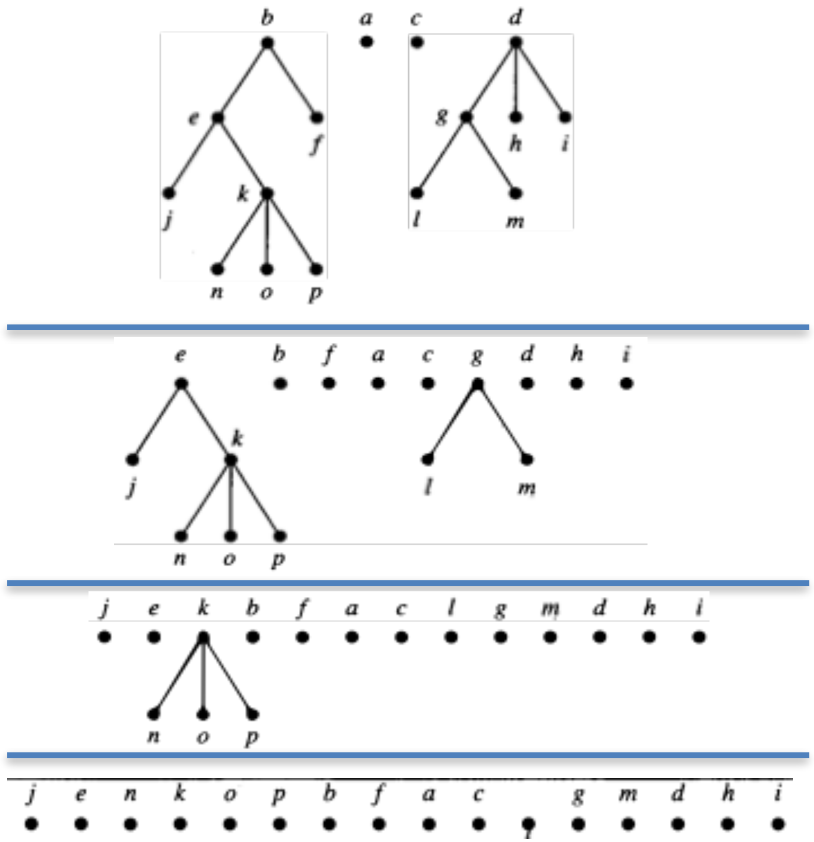
DEFINITION

Let T be an ordered binary 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.

EXEMPLE:



Inorder traversal :
Visit left most subtree, visit root, visit other subtrees left to right



3. Tree Traversal.

ALGORITHM Inorder Traversal.
procedure inorder(T : ordered rooted tree)
r:= root of T
if r is a leaf **then** list r
else
begin
    v:= first child of r from left to right
    T(v) := subtree with v as its root
    inorder(T(v))
    list r
    for each child c of r except for v from left to right
T(c) := subtree with c as its root
        inorder(T(c))
end

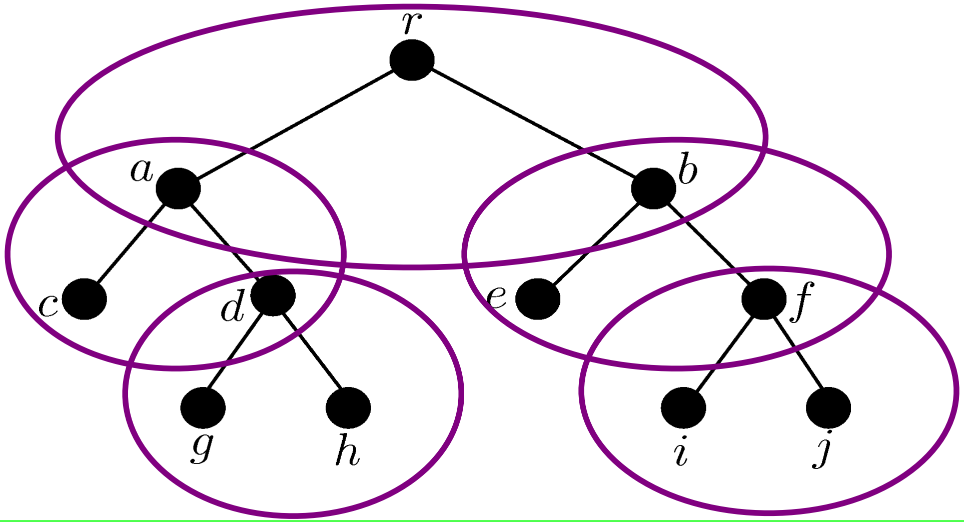
EXAMPLE:

$T_r \equiv \underline{T_a\ r\ T_b}$

$\equiv \underline{T_c\ a\ T_d}\ r\ \underline{T_e\ b\ T_f}$

$\equiv c\ a\ \underline{T_g\ d\ T_h}\ r\ e\ b\ \underline{T_i\ f\ T_j}$

$\equiv c\ a\ g\ d\ h\ r\ e\ b\ i\ f\ j$



3. Tree Traversal.

We now consider an application of a rooted tree in the study of computer science.

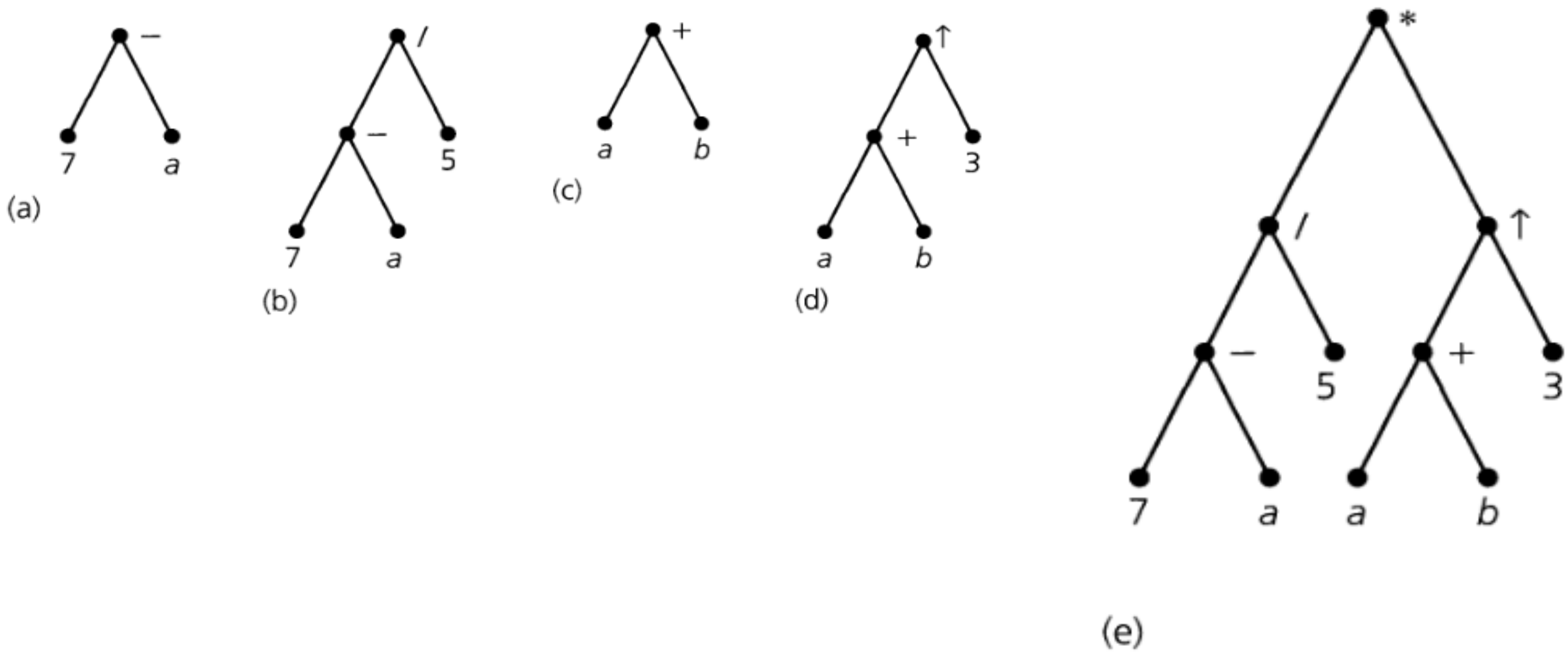
Infix, Prefix, and Postfix Notation

We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees. For instance, consider the representation of an arithmetic expression involving the operators + (addition), – (subtraction), * (multiplication), / (division), and ↑ (exponentiation). We will use parentheses to indicate the order of the operations. An ordered rooted tree can be used to represent such expressions, where 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).

3. Tree Traversal.

EXAMPLE: Construct the binary rooted tree for the algebraic expression

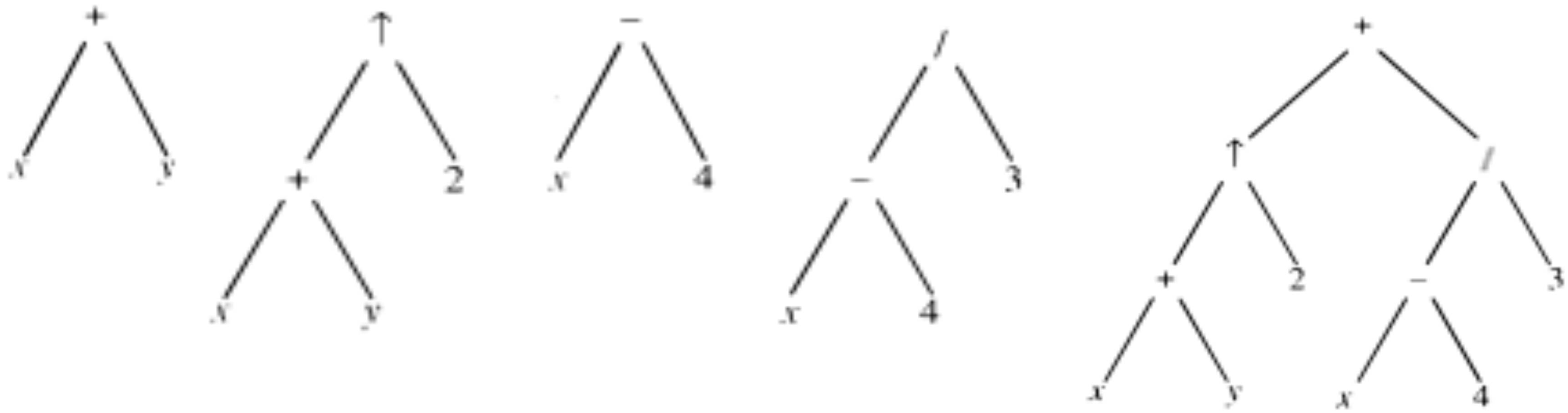
$$((7 - a)/5) * ((a + b) \uparrow 3),$$



3. Tree Traversal.

EXAMPLE: What is the ordered rooted tree that represents the expression?

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

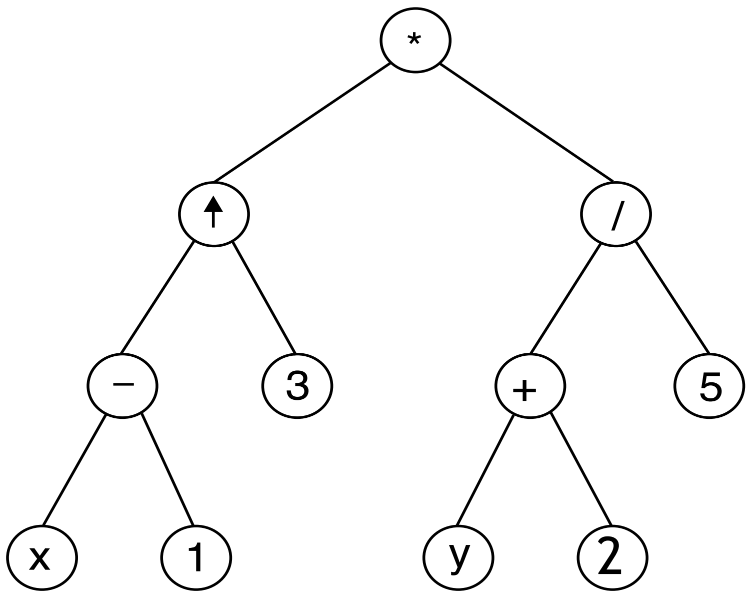


- The binary tree for this expression can be built from the bottom up.
- First, a subtree for the expression $x + y$ is constructed.
- Then this is incorporated as part of the larger subtree representing $(x + y) \uparrow 2$.
- Also, a subtree for $x - 4$ is constructed,
- and then this is incorporated into a subtree representing $(x - 4)/3$.
- Finally the subtrees representing $(x + y) \uparrow 2$ and $(x - 4)/3$ are combined.

3. Tree Traversal.

EXAMPLE: What is the ordered rooted tree that represents the expression?

$$(x-1)^3 * \frac{(y+2)}{5}.$$

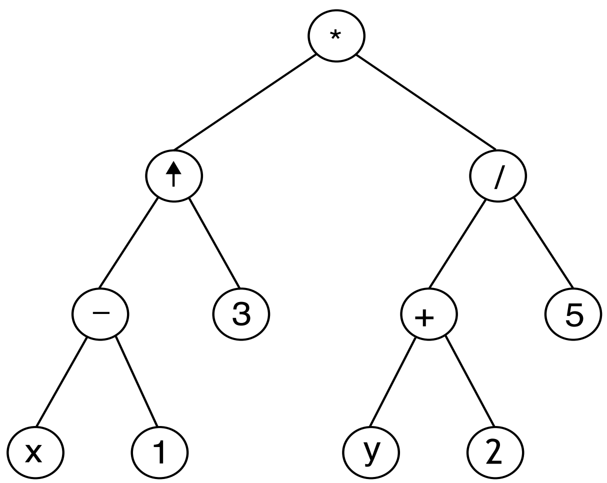


Prefix Notation

We obtain the **prefix notation (Polish notation)** of an arithmetic expression when we traverse its rooted tree in preorder.

EXAMPLE:

$$(x-1)^3 * \frac{(y+2)}{5}.$$



$T_* \equiv * \; T_\uparrow \; T_/$

$\equiv * \uparrow T_- \; T_3 \; / \; T_+ T_5$

$\equiv * \uparrow - \; T_x \; T_1 \; 3 \; / \; + \; T_y \; T_2 5$

$\equiv * \uparrow - \; x \; 1 \; 3 \; / \; + \; y \; 2 \; 5$

$$\boxed{* \uparrow - x \; 1 \;\; 3 \; / \; + \; y \; 2 \; 5.}$$

prefix notation (Polish notation)

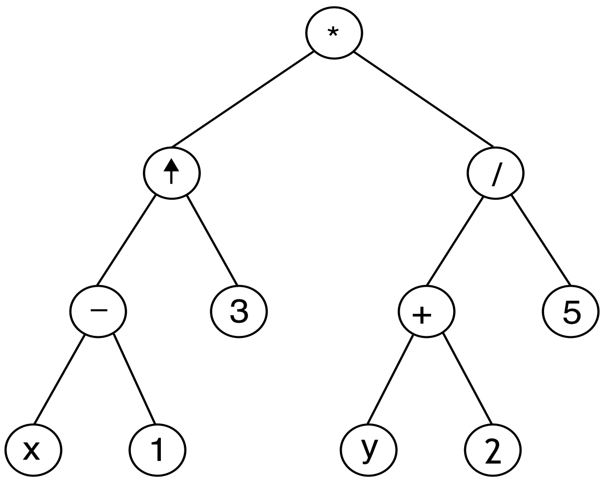
3. Tree Traversal.

Postfix Notation

We obtain the **postfix notation** (**reverse Polish notation**) of an arithmetic expression by traversing its binary tree in postorder.

EXAMPLE:

$$(x-1)^3 * \frac{(y+2)}{5}.$$



$T_* \equiv T_\uparrow \; T_/ \; *$

$\equiv T_- \; T_3 \uparrow T_+ T_5 \; / \; *$

$\equiv T_x \; T_1 - 3 \uparrow T_y \; T_2 + 5 \; / \; *$

$\equiv x \; 1 - 3 \uparrow y \; 2 + 5 \; / \; *$

$$\boxed{x\ 1\ -\ 3\ \uparrow\ y\ 2\ +\ 5\ /\ *.}$$

postfix notation (reverse Polish notation)

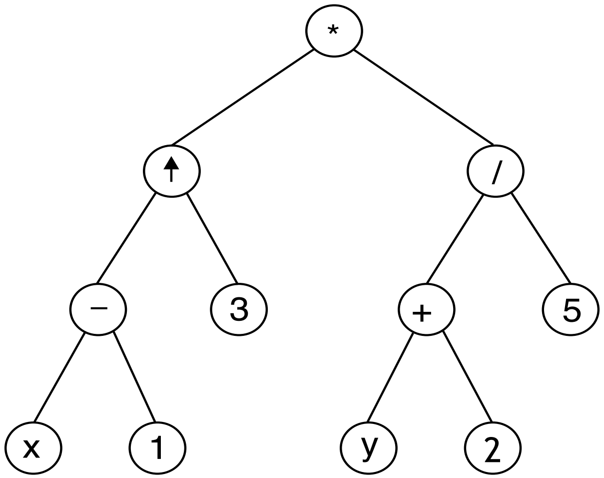
3. Tree Traversal.

Infix Notation

We obtain the **infix notation** of an arithmetic expression by traversing its binary tree in inorder **including parentheses in the inorder traversal whenever we encounter an operation**.

EXAMPLE:

$$(x-1)^3 * \frac{(y+2)}{5}.$$



$$\begin{aligned} T_* &\equiv T_{\uparrow} * T_/ \\ &\equiv T_- \uparrow T_3 * T_+ / T_5 \\ &\equiv T_x - T_1 \uparrow 3 * T_y + T_2 / 5 \\ &\equiv x - 1 \uparrow 3 * y + 2 / 5 \end{aligned}$$

$$\boxed{x \;-\; 1 \;\uparrow 3 * y + 2 / 5}$$

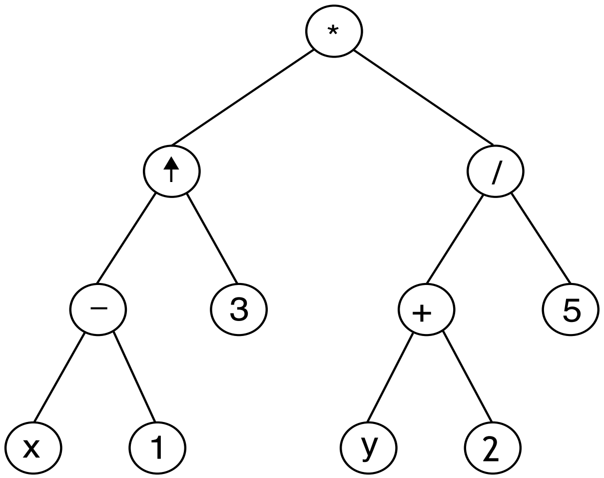
Inorder traversal AMBIGUOUS

3. Tree Traversal.

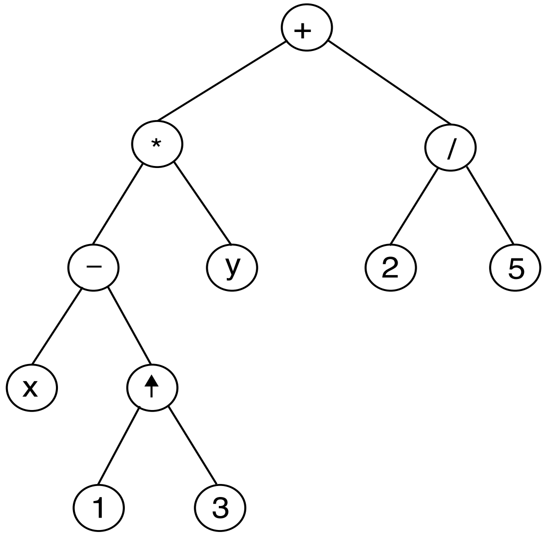
EXAMPLE:

Inorder traversals of different binary trees, which represent different expressions, may lead to the same inorder expression:

$$(x-1)^3 * \frac{(y+2)}{5}$$



$$(x-1^3) * y + \frac{2}{5}$$



$$x \; - \; 1 \; \uparrow 3 \; * \; y \; + \; 2 \; / \; 5$$

Inorder traversal
AMBIGUOUS

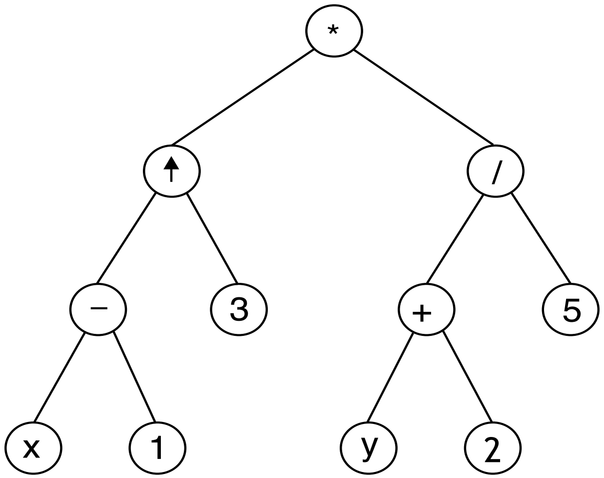
3. Tree Traversal.

Infix Notation

We obtain the **infix notation** of an arithmetic expression by traversing its binary tree in inorder **including parentheses in the inorder traversal whenever we encounter an operation**.

EXAMPLE:

$$(x-1)^3 * \frac{(y+2)}{5}$$



$T_* \equiv (T_\uparrow * T_/)$

$\equiv ((T_- \uparrow T_3) * (T_+ / T_5))$

$\equiv (((T_x - T_1) \uparrow 3) * ((T_y + T_2) / 5))$

$\equiv (((x - 1) \uparrow 3) *((y + 2) / 5))$

Infix notation
To make expressions unambiguous it is necessary to include parentheses.

Expressions in **Polish notation** or **reverse Polish notation** are unambiguous, so parentheses are not needed.

3. Tree Traversal.

EXAMPLE: Evaluating a Postfix Expression.

What is the value of the postfix expression 3 3 4 5 1 – * + + ?

$$3\ 3\ 4\ 5\ 1\ -\ \star\ +\ +$$

$$3\ 3\ 4\ \underbrace{5\ 1\ -}\ \star\ +\ +$$

$$3\ 3\ 4\ (5-1)\ \star\ +\ +$$

$$3\ 3\ \underbrace{4\ 4\ \star}\ +\ +$$

$$3\ 3\ (4\star 4)\ +\ +$$

$$3\ \underbrace{3\ 16\ +}\ +$$

$$3\ (3+16)\ +$$

$$\underbrace{3\ 19\ +}$$

$$(3+19)$$

$$22$$

$$3\ 3\ 4\ 5\ 1\ -\ \star\ +\ +\ =\ 22$$

EXAMPLE: Evaluating a Prefix Expression.

What is the value of the prefix expression – * 3 ↑ 5 2 2 ?

$$- \star 3 \uparrow 5\ 2\ 2$$

$$- \star 3 \underbrace{\uparrow 5\ 2}\ 2$$

$$- \star 3\ (5 \uparrow 2)\ 2$$

$$- \underbrace{\star\ 3\ 25}\ 2$$

$$- (3 \star 25)\ 2$$

$$\underbrace{-\ 75\ 2}$$

$$(75 - 2)$$

$$73$$

3. Tree Traversal.

EXAMPLE: Evaluating a Postfix Expression.

What is the algebraic expression corresponding to the postfix expression a b – 5 / x 3 – 4 ↑ x y * ↑ / ?
Determine the corresponding prefix expression.

a b - 5 / x 3 - 4 ↑ x y * ↑ /

(a - b) 5 / x 3 - 4 ↑ x y * ↑ /

((a - b) / 5) x 3 - 4 ↑ x y * ↑ /

((a - b) / 5) (x - 3) 4 ↑ x y * ↑ /

((a - b) / 5) ((x - 3) ↑ 4) x y * ↑ /

(((a - b) / 5) ((x - 3) ↑ 4)) (x * y) ↑ /

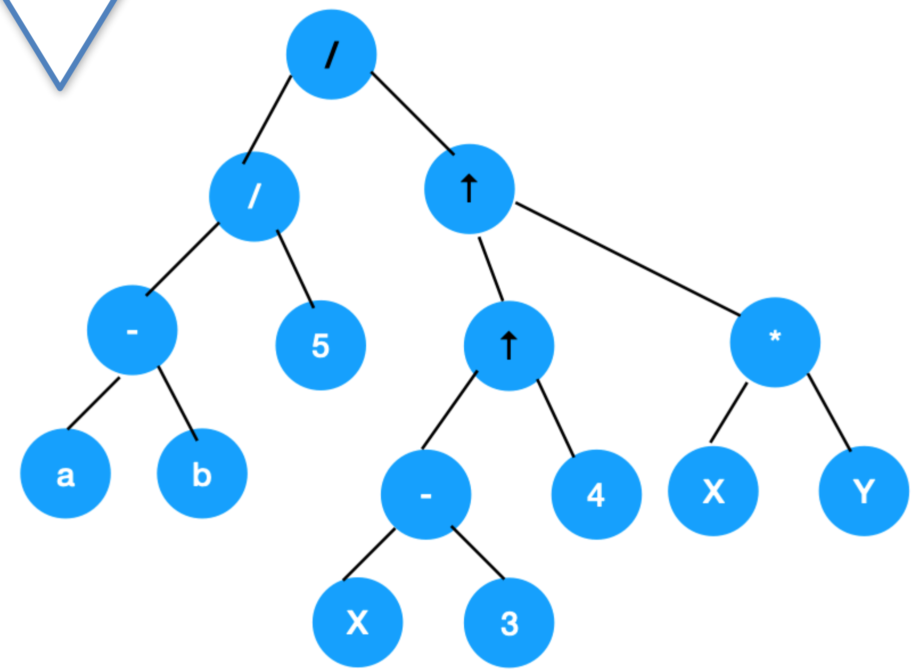
(((a - b) / 5) (((x - 3) ↑ 4)) ↑ (x * y)) /

(((a - b) / 5) / (((x - 3) ↑ 4)) ↑ (x * y))

Algebraic Expression

$$\frac{\left(\frac{a-b}{5}\right)}{\left((x-3)^4\right)^{(x*y)}}$$

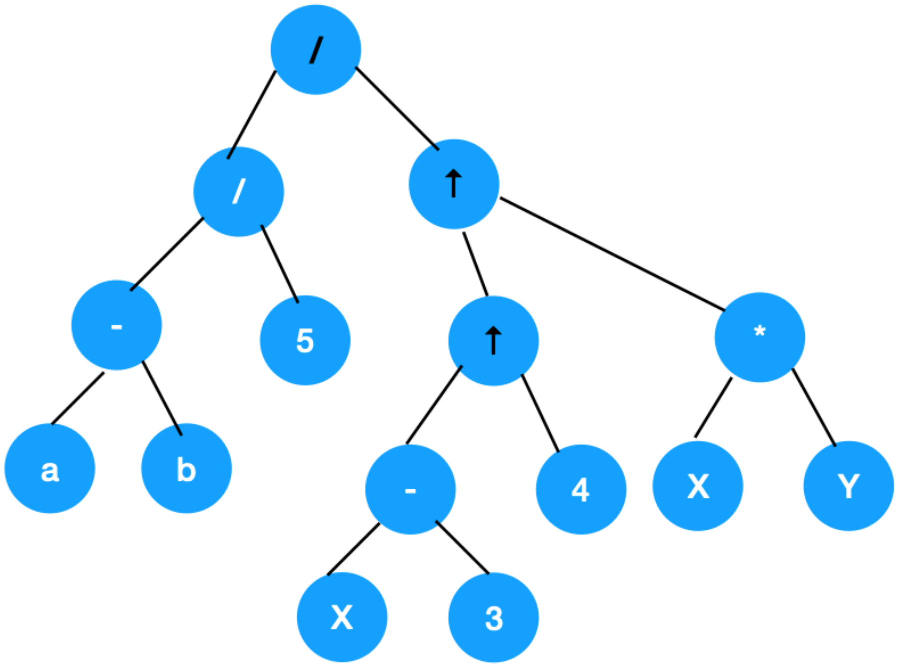
Tree



3. Tree Traversal.

EXAMPLE: Evaluating a Postfix Expression.

What is the algebraic expression corresponding to the postfix expression a b – 5 / x 3 – 4 ↑ x y * ↑ / ?
Determine the corresponding prefix expression.



From the tree we obtain the prefix expression

$$T_{/} \equiv \; / \; T_{/} \; T_{\uparrow}$$
$$\equiv \; // \; T_{-} \; T_{5} \uparrow T_{\uparrow} \; T_{*}$$
$$\equiv \; // \; - \; T_{a} \; T_{b} \; 5 \uparrow \uparrow T_{-} \; T_{4} \; {*}T_{x} \; T_{y}$$
$$\equiv \; // \; - \; a \; b \; 5 \uparrow \uparrow - \; T_{x} \; T_{3} \; 4{*} \; x \; y$$
$$\equiv \; // \; - \; a \; b \; 5 \uparrow \uparrow - \; x \; 3 \; 4 \; * \; x \; y$$