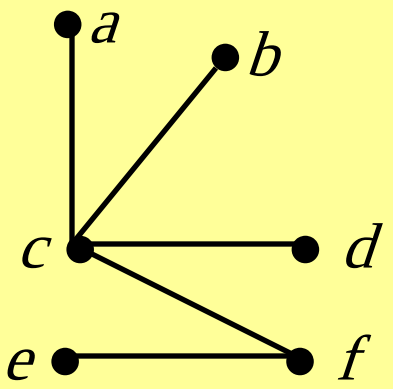
Chapter 10: Trees

- A **tree** is a connected simple undirected graph with no simple circuits.

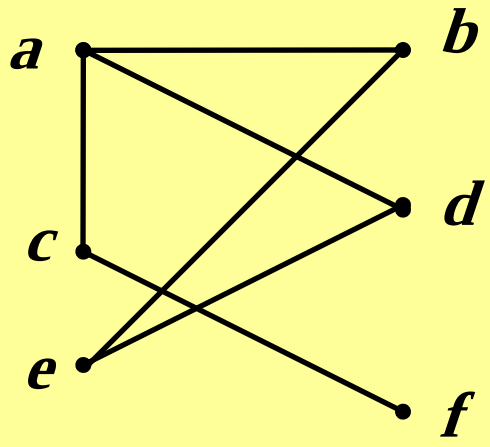
- **Properties:**
  - There is a unique simple path between any 2 of its vertices.
  - No loops.
  - No multiple edges.

Example 1



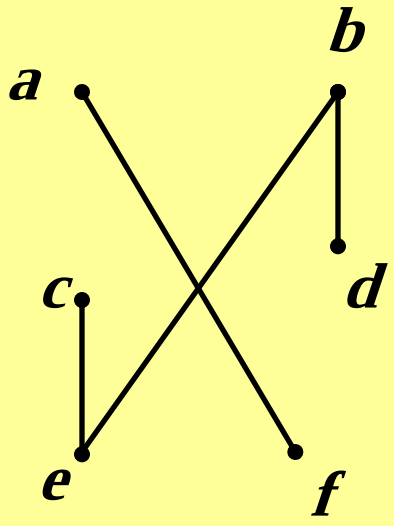
G1 : This graph is a <u>Tree</u> because it is a connected graph with no simple circuits

Example 2



G2: is <u>not a tree</u> " because there is a cycle *a, b, e, d, a*"

Example 3



G3: is <u>not a tree</u> "because it's not connected". In this case it's called forest in which each connected component is a tree.
Component 1: *a, f*
Component 2: *c, e, b, d*

Forest

- An undirected graph without simple circuits is called a **forest**.

– You can think of it as a set of trees having disjoint sets of nodes.

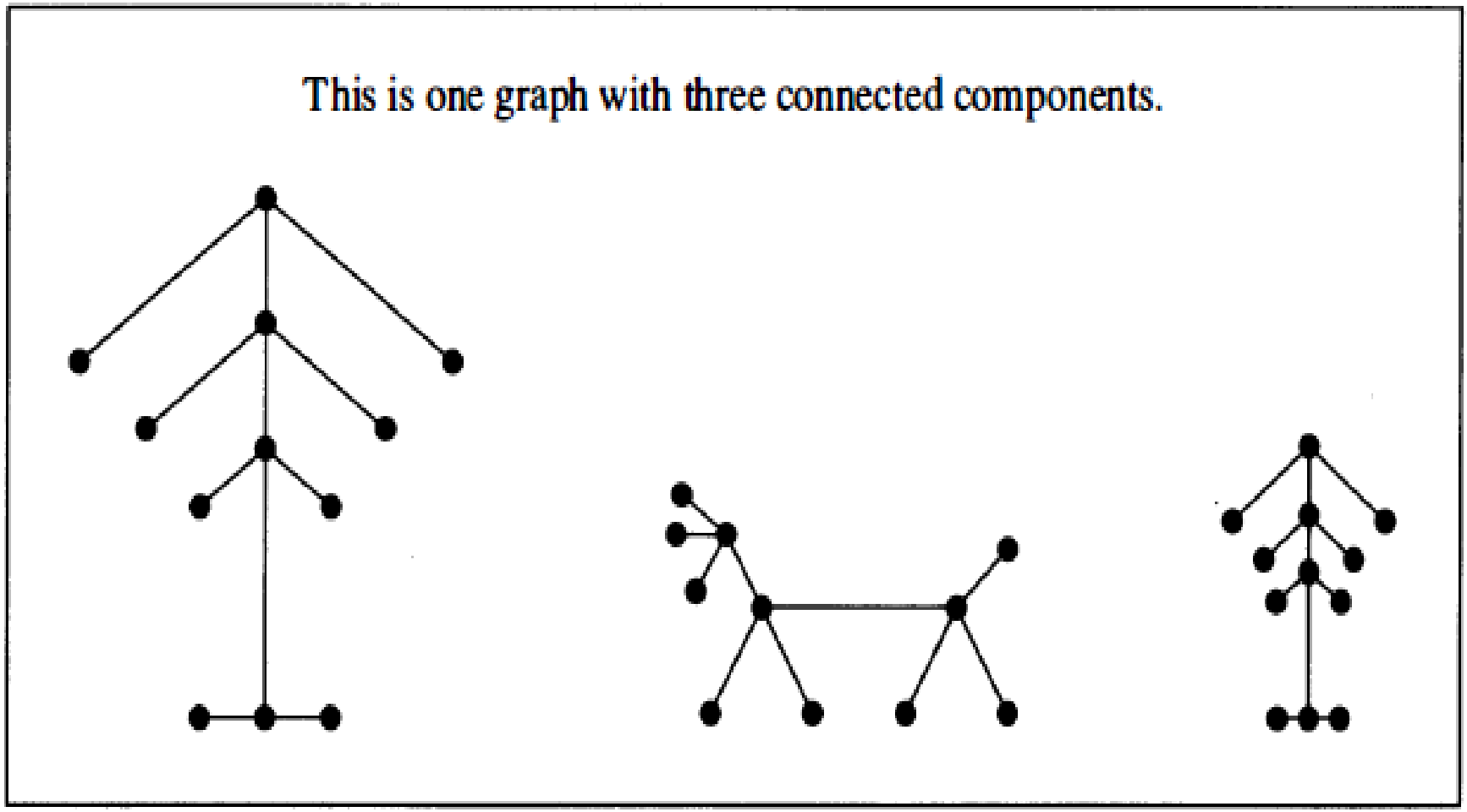
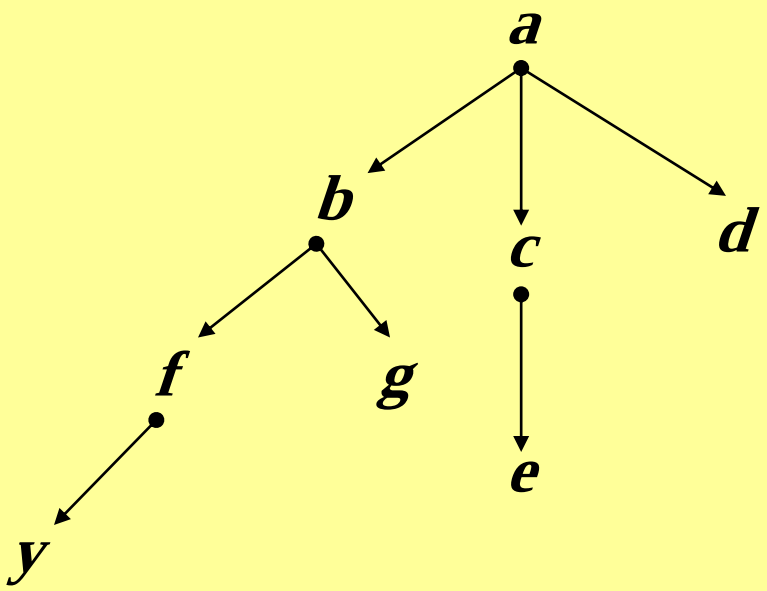


FIGURE 3 Example of a Forest.

Rooted (Directed) Trees

- A **rooted tree** is a tree in which one node has been designated the **root** and every edge is directed away from the root.
- You should know the following terms about rooted trees:
  - Root, Parent, Child, Siblings, Ancestors, Descendents, Leaf, Internal node, Subtree.

Definitions



- **Root**: Vertex with in-degree 0

[Node *a* is the root]

Definitions

- **Parent**: Vertex *u* is a parent, such that there is directed edge from *u* to *v*.

  [*b* is parent of *g* and *f*]
- **Child:** If *u* is parent of *v*, then *v* is child of *u*.

  [*g* and *f* are children of *b*]
- **Siblings:** Vertices with the same parents.

  [*f* and *g*]
- **Ancestors:** Vertices in path from the root to vertex *v*, excluding *v* itself, including the root.

  [Ancestors of *g*: *b*, *a*]

Definitions

- **Descendents**: All vertices that have v as ancestors.

  [**Descendents of *b* : *f, g, y*]**

- **Leaf:** Vertex with no children.

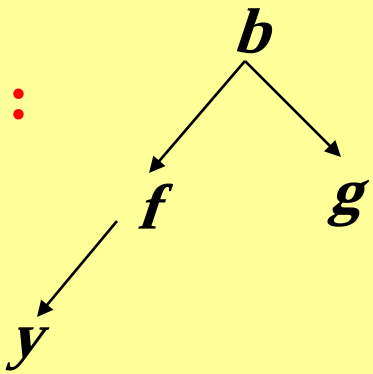
  [*y, g, e, d*]

- **Internal vertices:** Vertices that have children.

  [*a, b, c, f*]

- **Subtree:** Subgraphs consisting of v and its descendents and their incident edges.

  Subtree rooted at *b* :



Definitions

- **Level (of *v*)** is length of unique path from root to *v*.

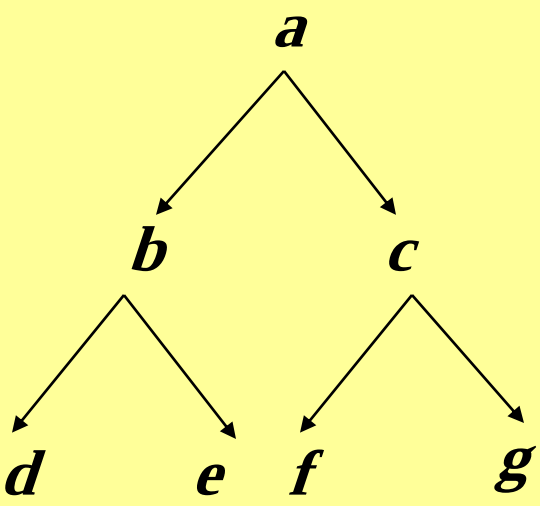
  [level of root = 0, level of *b* = 1, level of *g* = 2]
- **Height** is maximum of vertices levels.

  [Height = 3]

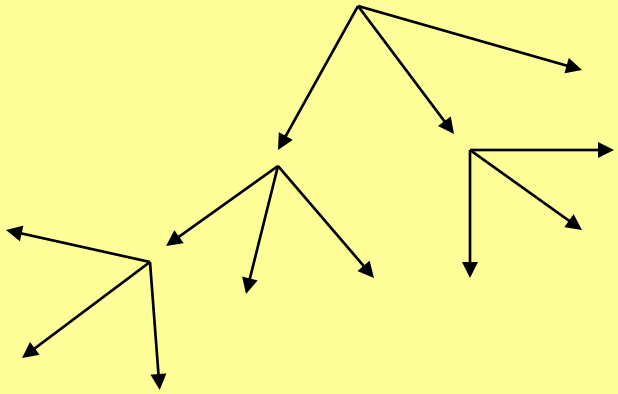
m-ary Trees

- A rooted tree is called ***m*-ary** if every internal vertex has no more than *m* children.
- It is called **full *m*-array** if every internal vertex has **exactly** *m* children.
- A 2-ary tree is called a **binary tree**.

Example



Full binary tree

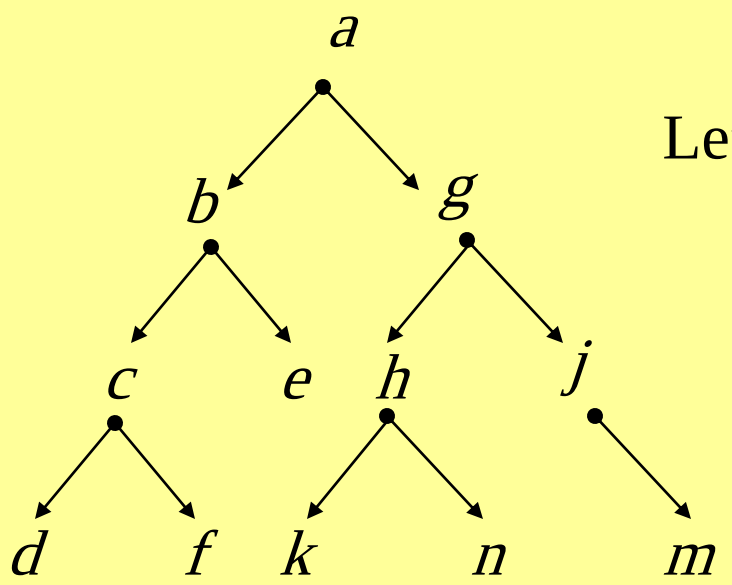


Full 3-ary tree

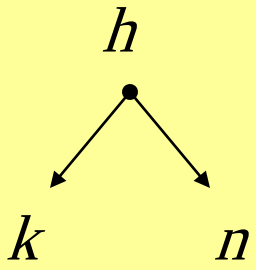
Ordered Rooted Tree

- A rooted tree where the children of each internal node are ordered.
- In ordered binary trees, we can define:
  - **left child, right child**
  - **left subtree, right subtree**
- For *m*-ary trees with $m > 2$, we can use terms like "leftmost", "rightmost," etc.

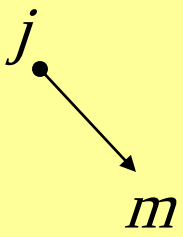
Examples



Left subtree of g



Right subtree of g



Left child of c is d , Right child of c is f

Trees as Models

Saturated Hydrocarbons and Trees.

Graphs can be used to represent molecules, where atoms are represented by vertices and bonds between them by edges.

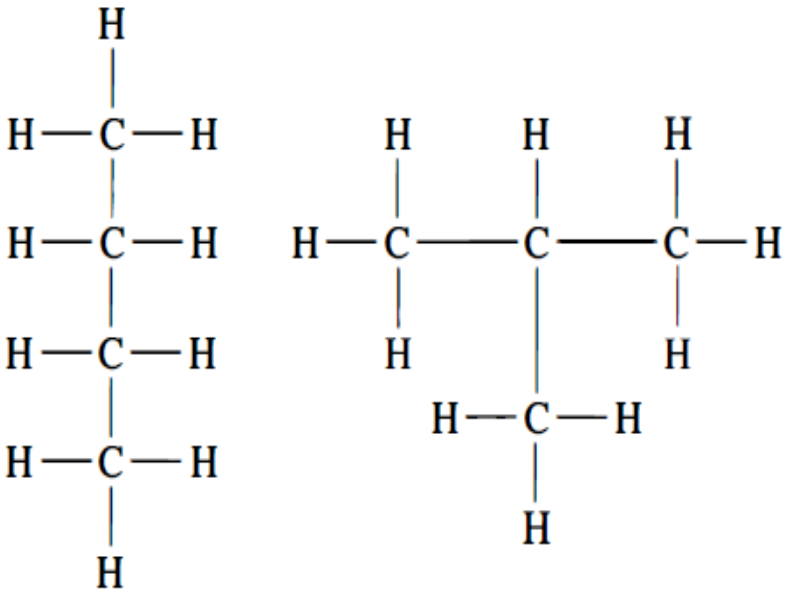


FIGURE 9 The Two Isomers of Butane.

Representing Organizations The structure of a large organization can be modeled using a rooted tree. Each vertex in this tree represents a position in the organization. An edge from one vertex to another indicates that the person represented by the initial vertex is the (direct) boss of the person represented by the terminal vertex.

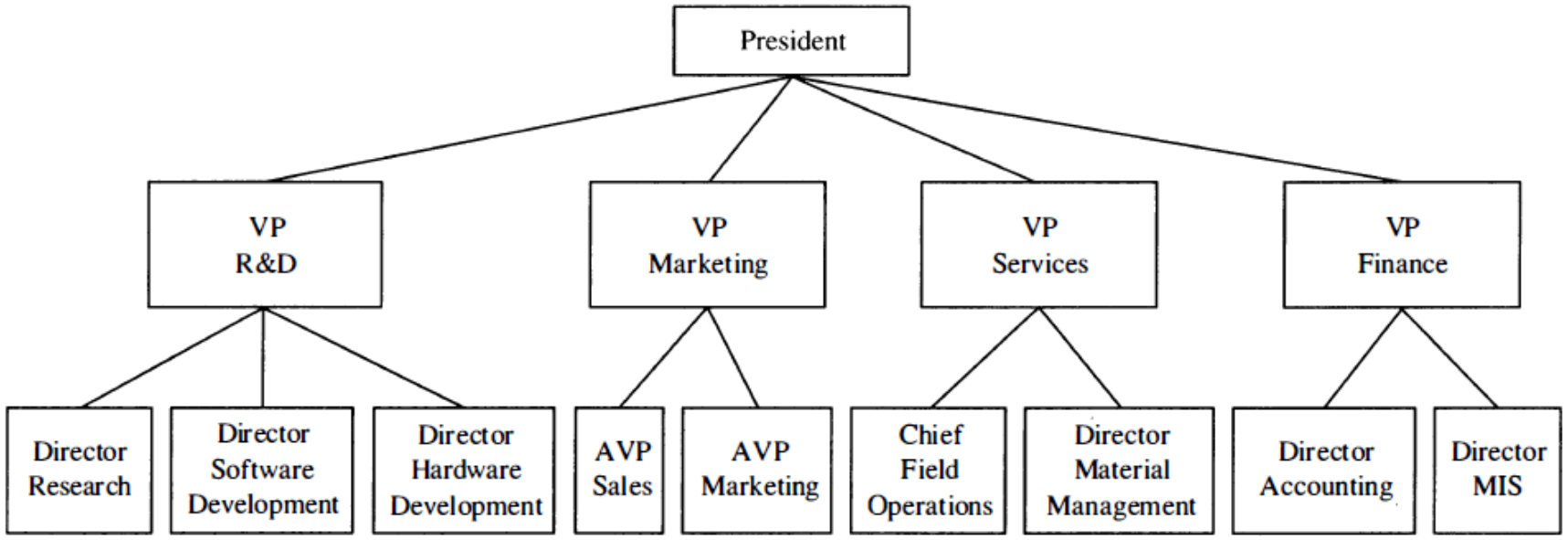


FIGURE 10 An Organizational Tree for a Computer Company.

Computer File Systems Files in computer memory can be organized into directories. A directory can contain both files and subdirectories. The root directory contains the entire file system. Thus, a file system may be represented by a rooted tree, where the root represents the root directory, internal vertices represent subdirectories, and leaves represent ordinary files or empty directories.

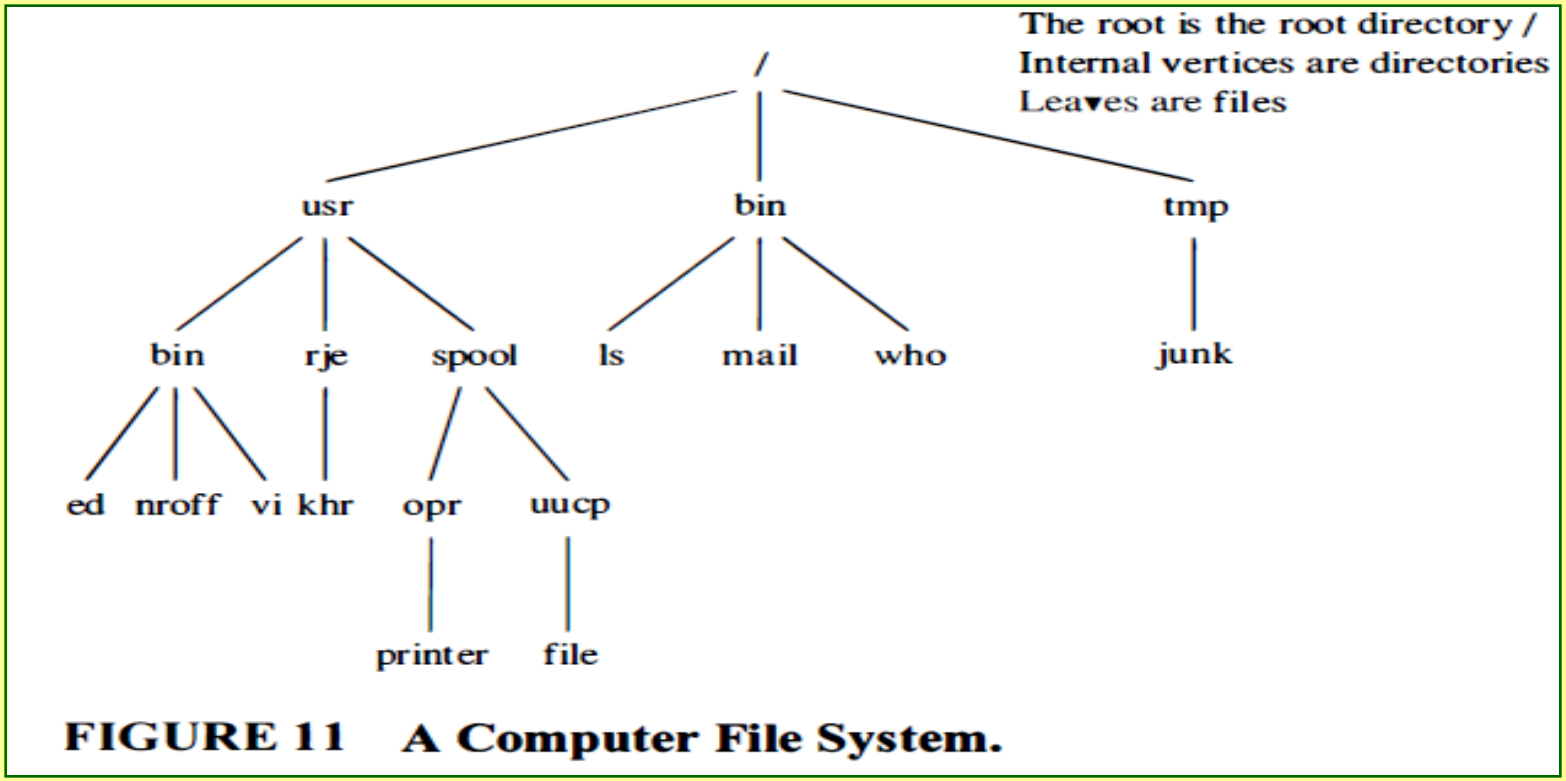


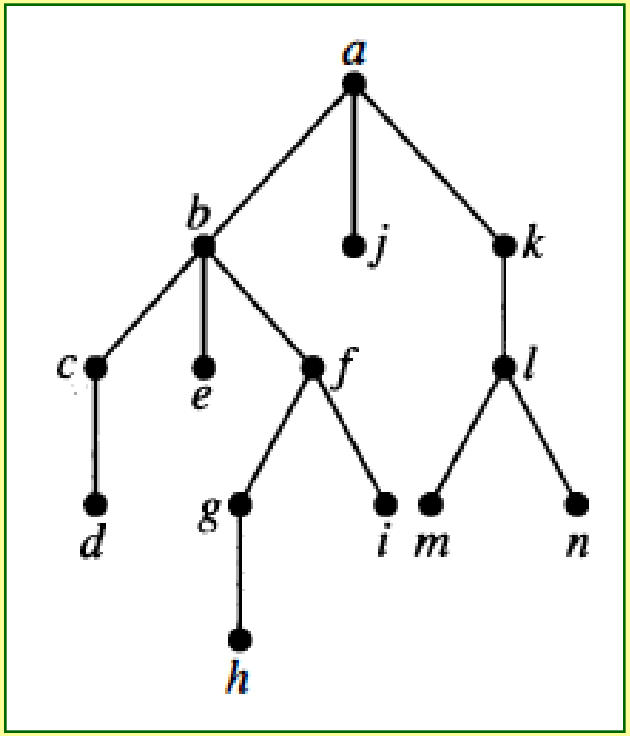
FIGURE 11 A Computer File System.

Properties of Trees

1- A tree with n vertices has

n - 1 edges.

e.g. The tree in the figure has
14 vertices and 13 edges

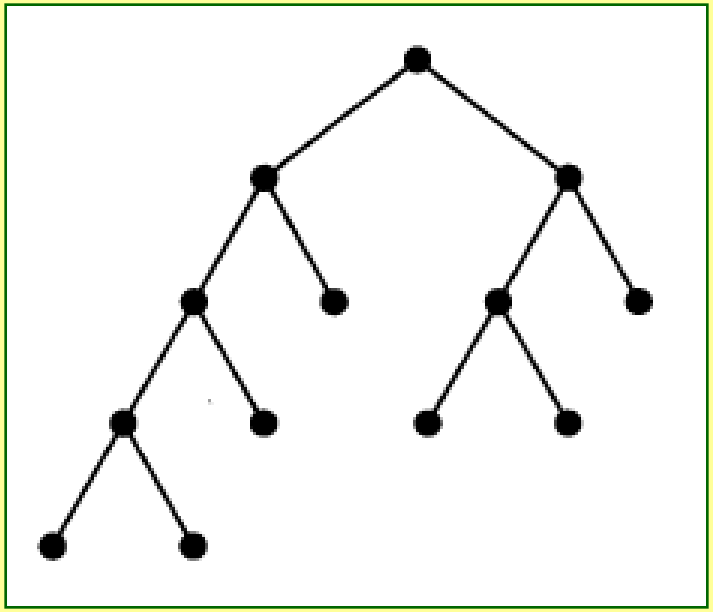


Properties of Trees

2- A full *m*-ary tree with *I* internal vertices and *L* leaves contains:

***n* = *m* × *I* + 1 vertices**

***n* = *I* + *L* vertices**



e.g. The full binary tree in the figure has:

Internal vertices $I = 6$

Leaves $L = 7$

Vertices $13 = (2)(6) + 1$

Summary

For a full *m*-ary tree:

(i) Given n vertices, $I = (n - 1) / m$ internal vertices and
$L = n - I = [(m - 1) \times n + 1] / m$ leaves.

(ii) Given I internal vertices, $n = m \times I + 1$ vertices and
$L = n - I = (m - 1) \times I + 1$ leaves.

(iii) Given L leaves, $n = (m \times L - 1) / (m - 1)$ vertices and
$I = n - L = (L - 1) / (m - 1)$ internal vertices.

In the previous example:

$m = 2$, $n = 13$, $I = 6$ and $L = 7$

(i) $I = (13 - 1) / 2 =$ **6** and $L = 13 - 6 =$ **7**

(ii) $n = 2 \times 6 + 1 =$ **13** and $L = 13 - 6 =$ **7**

(iii) $n = (2 \times 7 - 1) / (2 - 1) =$ **13** and $I = 13 - 7 =$ **6**

Properties of Trees

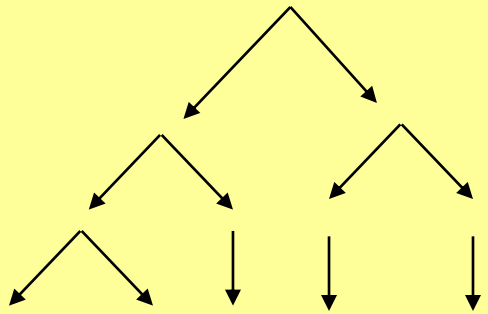
3- The **level** of a vertex in a rooted tree is the length of the path from the root to the vertex (level of the root is 0)

4- The **height** of the rooted tree is the maximum of the levels of vertices (length of the longest path from the root to any vertex)

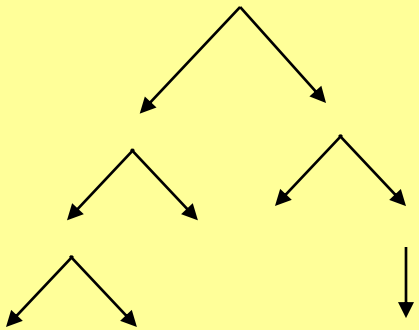
Balanced Trees

- **Balanced Tree**

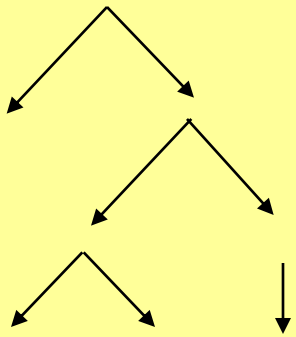
  A rooted ***m*-ary** tree of height ***h*** is balanced if all **leaves** are at levels ***h*** or ***h* - 1**.



Balanced



Balanced



Not Balanced

10.2 Applications of Trees

- **Binary Search Trees**

  Searching for items in a list is one of the most important tasks that arises in computer science. Our primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered. This can be accomplished through the use of a **binary search tree.**

Binary Search Trees

A **binary search tree** is a binary tree in which each child of a vertex is designated as a right or left child, no vertex has more than one right child or left child, and each vertex is labeled with a key, which is one of the items. Furthermore, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

Example

Form a binary search tree for the words mathematics, physics, geography, zoology, meteorology, geology, psychology, and chemistry (using alphabetical order).

Mathematics, physics, geography, zoology, meteorology, geology, psychology, and chemistry

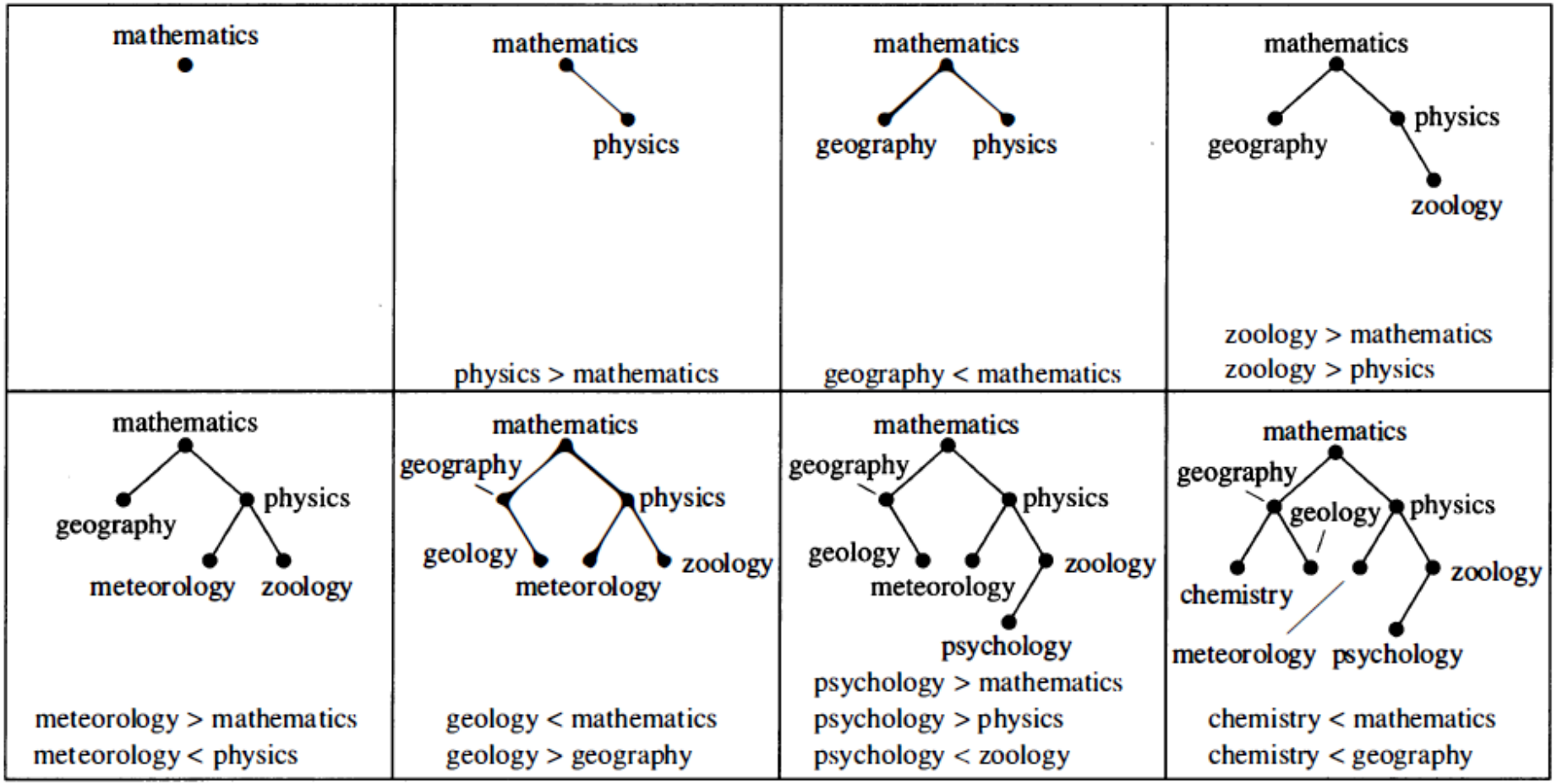


FIGURE 1 Constructing a Binary Search Tree.

Decision Trees

Rooted trees can be used to model problems in which a series of decisions leads to a solution. For instance, a binary search tree can be used to locate items based on a series of comparisons, where each comparison tells us whether we have located the item, or whether we should go right or left in a subtree. A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a **decision tree**. The possible solutions of the problem correspond to the paths to the leaves of this rooted tree.

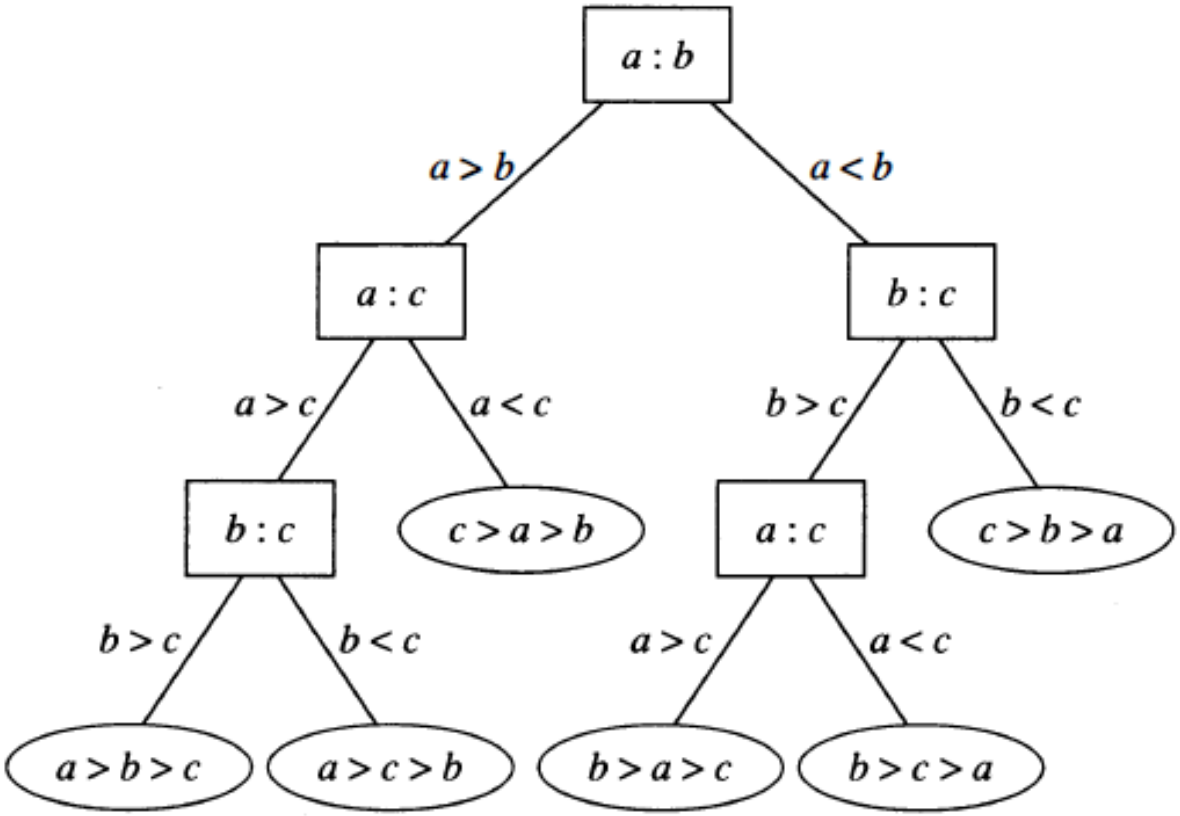


FIGURE 4 A Decision Tree for Sorting Three Distinct Elements.

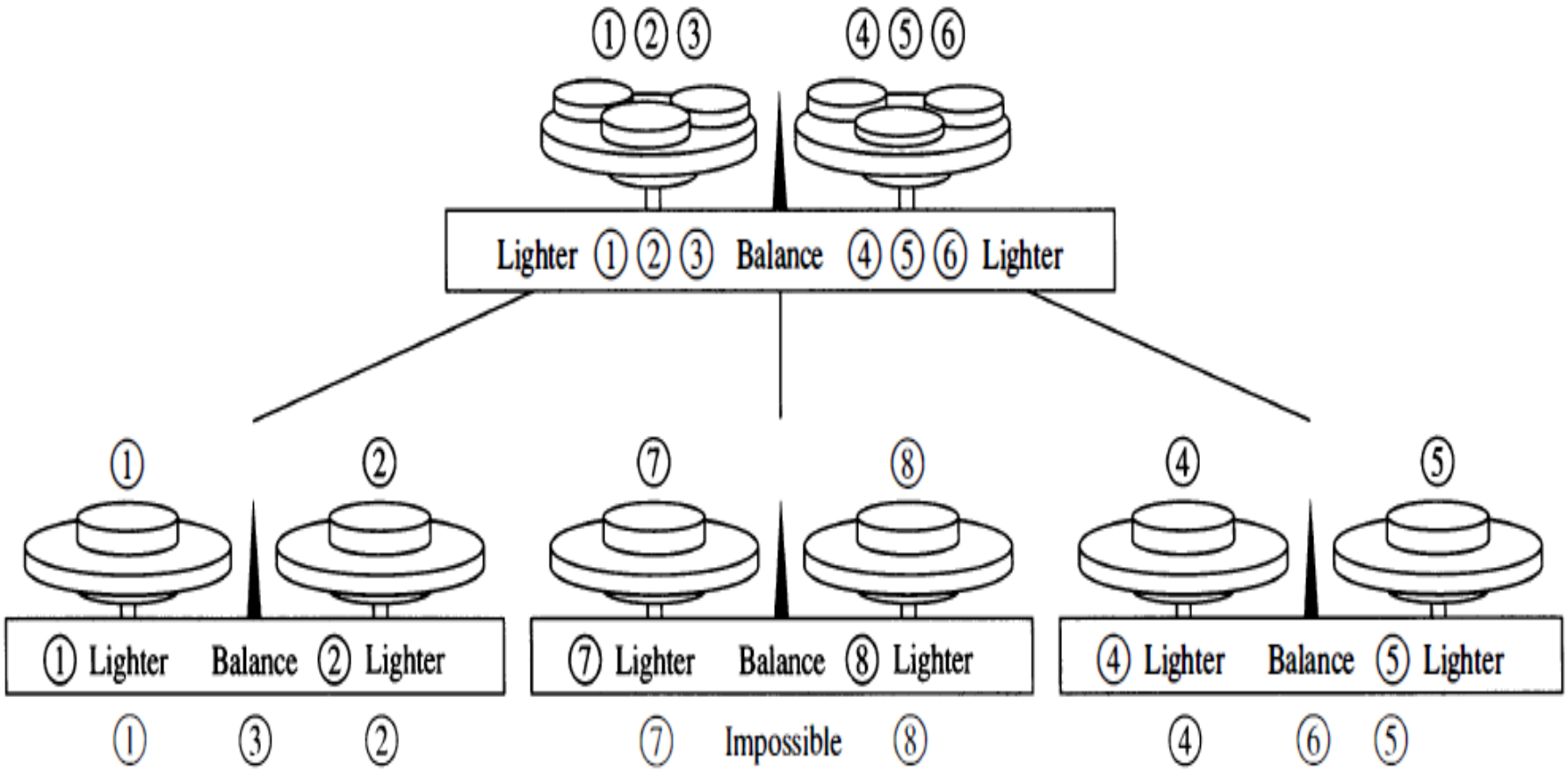


FIGURE 3 A Decision Tree for Locating a Counterfeit Coin. The counterfeit coin is shown in color below each final weighing.

Prefix Codes

Consider using bit strings of different lengths to encode letters. When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end. One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter. Codes with this property are called **prefix codes**.

Example

The tree in the figure represents the encoding of

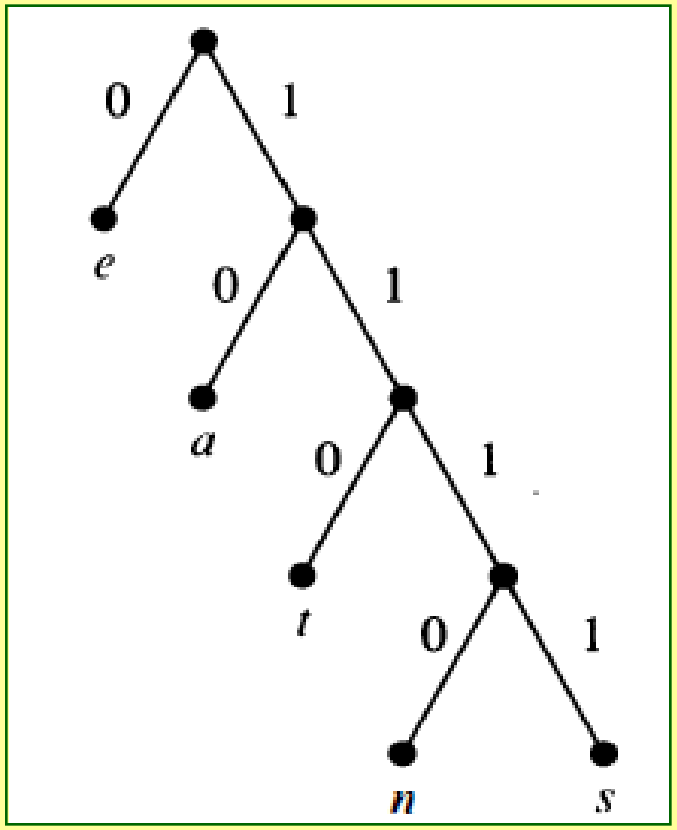
e by 0,

a by 10,

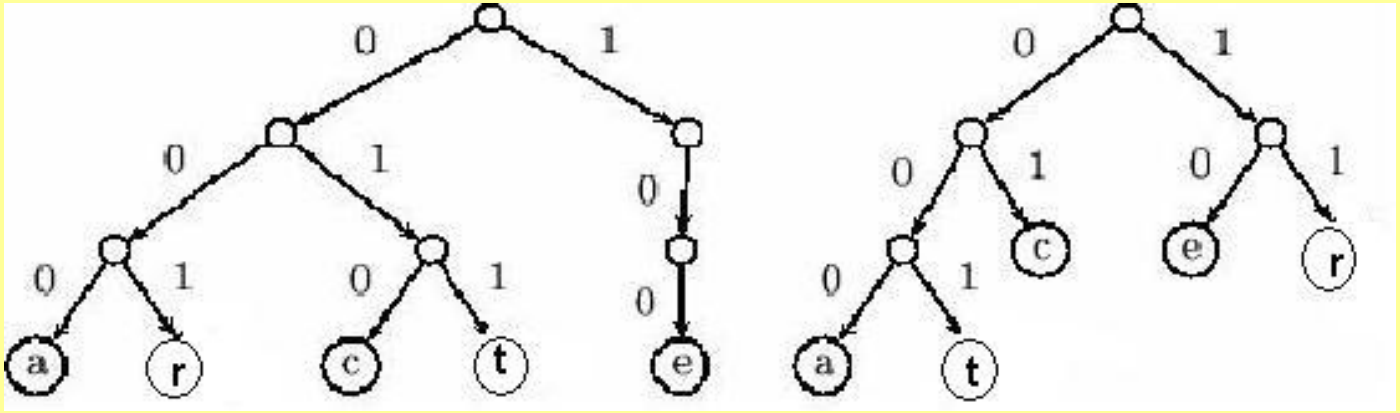
t by 110,

n by 1110,

s by 1111.



Huffman Codes



- On the left tree the word **rate** is encoded

  001 000 011 100

- On the right tree, the same word **rate** is encoded

  11 000 001 10

10.3 Tree Traversal

- Traversal algorithms
  - Pre-order traversal
  - In-order traversal
  - Post-order traversal
- Prefix / Infix / Postfix notation

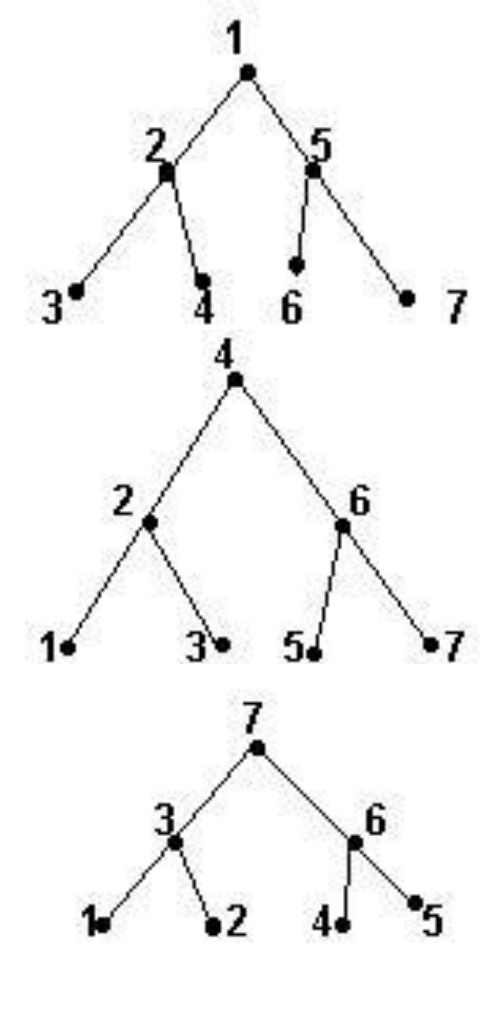
Traversal Algorithms

Is visiting every vertex of ordered rooted tree.

- **Pre-order:** **R**oot, **L**eft, **R**ight.
- **In-order:** **L**eft, **R**oot, **R**ight.
- **Post-order:** **L**eft, **R**ight, **R**oot.

Tree Traversals

- Pre-order traversal
- In-order traversal
- Post-order traversal

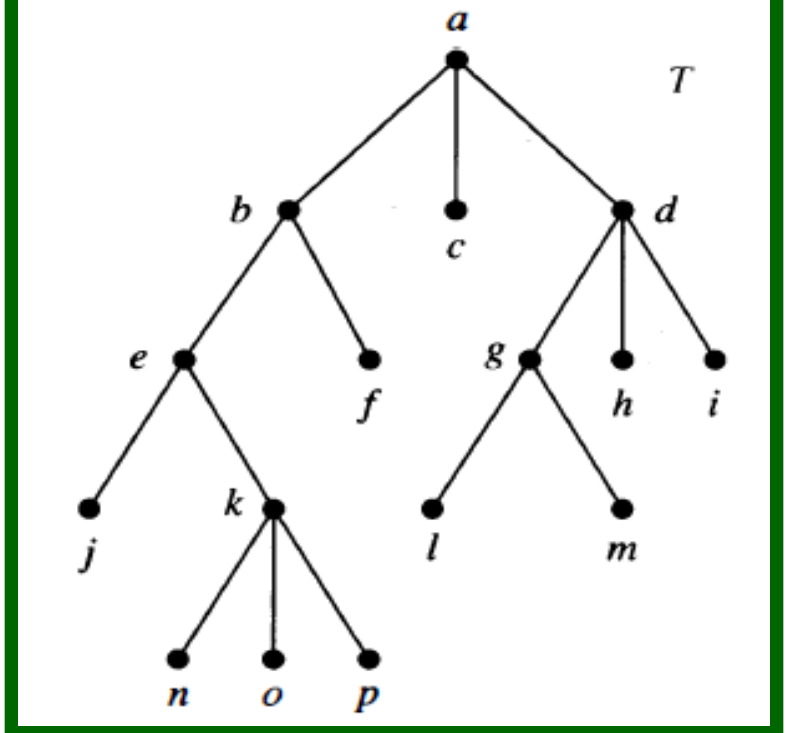


Tree Traversals

Pre-order: **R**oot, **L**eft, **R**ight.

In-order: **L**eft, **R**oot, **R**ight.

Post-order: **L**eft, **R**ight, **R**oot.



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

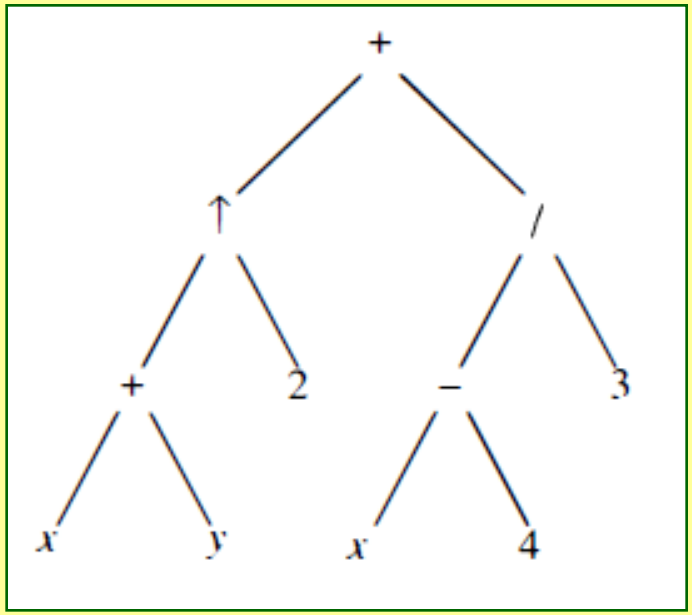
In-order: j e n k o p b f a c l g m d h i

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

Infix, Prefix, and Postfix Notation

A tree can be used to represents mathematical expressions.

Example: **((*x* + *y*)^2) + ((*x* – 4) / 3)**



Prefix: + ^ + *x y* 2 / – *x* 4 3

Postfix: *x y* + 2 ^ *x* 4 – 3 / +

Prefix Codes

Infix: In-order traversal of tree must be fully parenthesized to remove ambiguity.

Example: ***x*** **+ 5 / 3 :** **(*x* + 5) / 3 ,** ***x*** **+ (5 / 3)**

Prefix (polish): Pre-order traversal of tree (no parenthesis needed)

Example: From the above tree → **+ * + *x y* 2 / – *x* 4 3**

Postfix: Post-order traversal (no parenthesis needed)

Example: From the above tree → ***x y*** **+ 2 * *x* 4 – 3 / +**

1. Evaluating a **Prefix** Expression: (**Pre-order**: **Right to left**)

+ − * 2 3 5 / ↑ 2 3 4

2 ↑ 3 = 8

+ − * 2 3 5 / 8 4

8 / 4 = 2

+ − * 2 3 5 2

2 3 = 6

+ − 6 5 2

6 − 5 = 1

+ 1 2

1 + 2 = 3

Value of expression 3

2. Evaluating a **Postfix** Expression: (**Post-order**: **Left to right)**