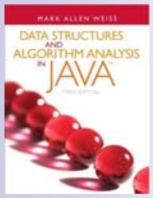
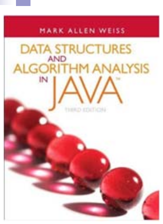


FLORIDA INTERNATIONAL UNIVERSITY

Trees
(I)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences



COP-3530 - Data Structures

Module #3: Trees (part I)

Outline:

- **Generic (rooted) trees.**
- **Some tree terms.**
- **Implementation of generic trees.**
- **Generic tree simple application.**
- **Tree traversals:**
  - **Post-order**
  - **Pre-order**

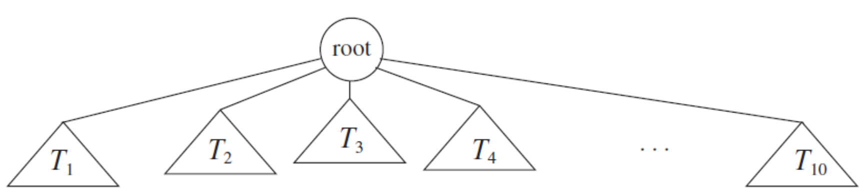
2

Nonlinear data structures. Motivation

- One of the disadvantages of using an **array** or **linked list** to store data is the time necessary to search for an item.
- Since both the arrays and Linked Lists are **linear structures** the time required to search a "linear" list is proportional to the size of the data set.
- For example, if the size of the data set is n, then the number of comparisons needed to find (or not find) an item may be as bad as some multiple of n. More efficient data structures are needed to store and search data.
- On the other hand there are **many situations in which information has a** hierarchical structure like that found in family trees or organization charts.
- In this module, the goal is to extend the concept of linear structure to a structure that may have multiple relations among its nodes. Such a structure is called a **tree**.

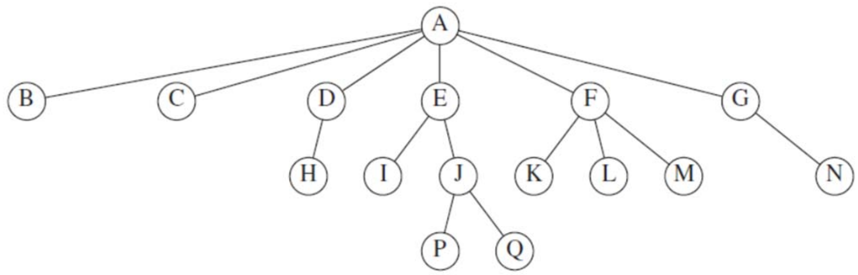
Preliminaries. Generic Trees

- **Definition (recursive definition):**
  - A **tree** is a collection of **nodes**.
  - The collection can be **empty**; otherwise, a **tree** consists of a special node r, called the **root**, and zero or more nonempty **(sub)trees** $T_1, T_2, \ldots, T_k$.
  - Each of whose roots are connected by a directed **edge** from r.
  - The root of each **subtree** is said to be a **child** of r, and r is the **parent** of each **subtree root**.

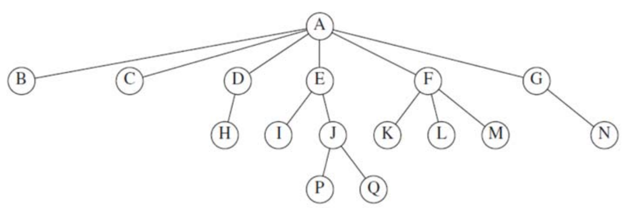


Generic Trees

- **Important points:**
  - **Tree** - collection of ***N*** **nodes**, one of which is the **root**, and ***N* − 1 edges**.
  - That there are ***N* − 1 edges** follows from the fact that each edge connects some node to its parent, and every node except the **root** has one parent

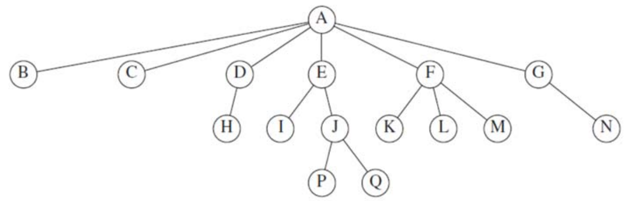


Some Tree terms



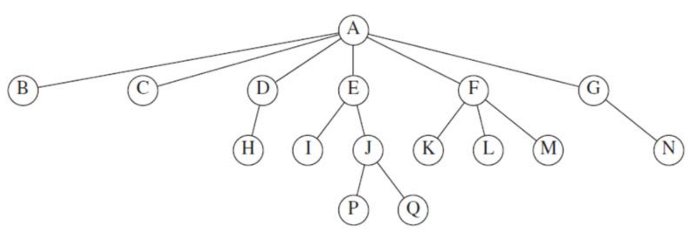
- **Leafs -** Nodes with no children (B,C,H,I,P,Q,K,L,M, and N).
- **Siblings** - Nodes with the same parent (Ex. K,L, and M are all siblings.
- **Grandparent & Grandchild** relations can be defined in a similar manner that siblings.
- A **path** from node n_1 to n_k is defined as a sequence of nodes n_1, n_2, . . . , n_k such that $\mathbf{n_i}$ **is the parent of** $\mathbf{n_{i+1}}$ for $1 \leq i < k$.
- The **length** of this path is the number of edges on the path, namely $k - 1$. There is a path of length zero from every node to itself.
- <u>**Notice that in a tree there is exactly one path from the root to each node**</u>.

Some Tree terms (cont…)



- For any node n_i, the **depth** of n_i is the **length** of the **unique path** from the root to n_i. Thus, the root is at depth 0.
- The **height** of n_i is the **length of the longest path** from n_i to a leaf. Thus all leaves are at height 0. The height of a tree is equal to the height of the root.
- **Example:**
  - E is at depth 1 and height 2; F is at depth 1 and height 1; the height of the tree is 3.
- **The depth of a tree is equal to the depth of the deepest leaf.**

Some Tree terms (cont…)



depth 0

depth 1

depth 2

depth 3

Nodes: 16

Edges: 15 (Nodes – 1)

Root: A

Leaves: B, C, H, I, P, Q, K, L, M, N

Height = 3

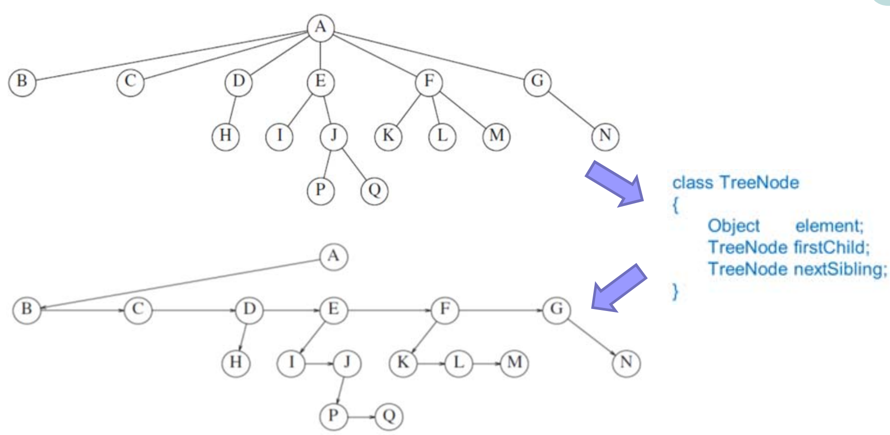
Implementation of generic trees

- **Trivial idea:**
  - To have each node, besides its data, a link to each child of the node.
- **But:**
  - The number of children per node can vary so greatly and is not known in advance!

- **Simple solution:**
  - Keep the children of each node in a linked list of tree nodes.

```
class TreeNode
{
      Object      element;
      TreeNode firstChild;
      TreeNode nextSibling;
}
```

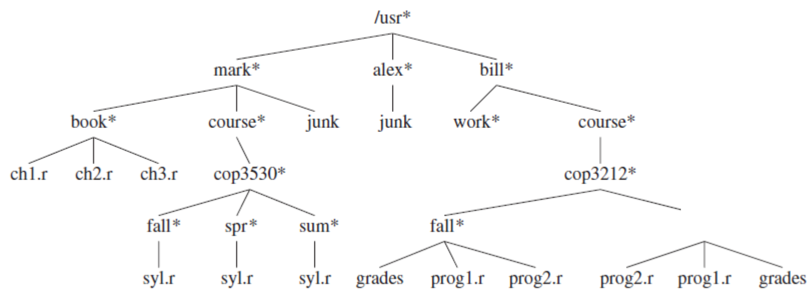

Implementation of Trees



- Arrows that point downward are **firstChild links**. Horizontal arrows are **nextSibling links**. Null links are not drawn, because there are too many.
- **Example:** Node E has both a link to a sibling (F) and a link to a child (I), while some nodes have neither.

Generic tree simple application

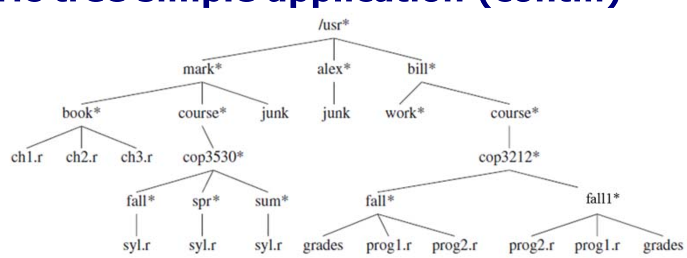
- **Example of application for trees:**
  - The directory structure in many common operating systems, including UNIX and DOS.



- The asterisk next to the name indicates that **/usr** is itself a directory.
- The filename **/usr/mark/book/ch1.r** is obtained by following the leftmost child three times. Each / after the first indicates an edge; the result is the full **pathname**.

11

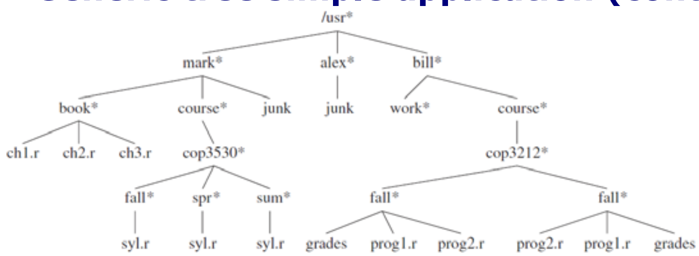
Generic tree simple application (cont…)



- If we need to list the names of all of the files in the directory. Our output format will be that files that are depth d_i will have their names indented by d_i tabs.

```
private void listAll( int depth )
{
    printName( depth ); // Print the name of the object
    If ( isDirectory( ) )
        for each file c in this directory (for each child)
            c.listAll( depth + 1 );
}
public void listAll( )
{
     listAll( 0 );
}
```

Generic tree simple application (cont…)




```
private void listAll( int depth )
{
    printName( depth ); // Print the name of the object
    if( isDirectory( ) )
       for each file c in this directory (for each child)
          c.listAll( depth + 1 );
}
public void listAll( )
{
     listAll( 0 );
}
```

- **Logic:** The name of the file object is printed out with the appropriate number of tabs. If the entry is a directory, then we process all children recursively, one by one. These children are one level deeper and thus need to be indented an extra space.

```
/usr
    mark
        book
            ch1.r
            ch2.r
            ch3.r
        course
            cop3530
                fall
                    syl.r
                spr
                    syl.r
                sum
                    syl.r
        junk
    alex
        junk
    bill
        work
        course
            cop3212
                fall
                    grades
                    prog1.r
                    prog2.r
                fall
                    prog2.r
                    prog1.r
                    grades
```

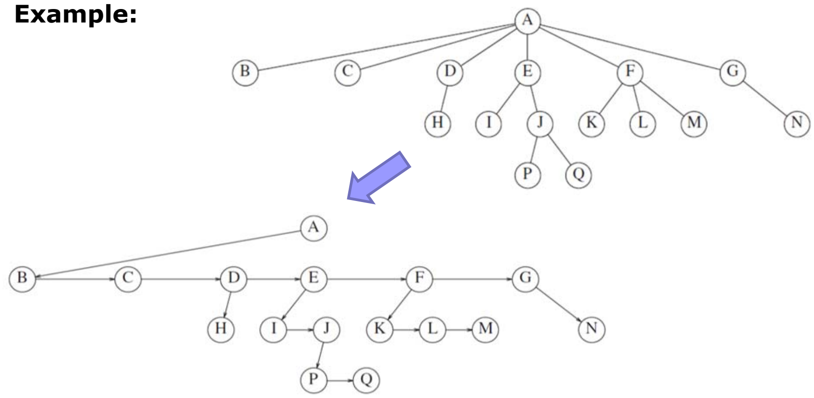
Tree Traversals

- **Tree traversal** is a process to visit all the nodes of a tree and may print their values too.
- Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.
- Because, all nodes are connected via edges (links) we always start from the root node. That is, we cannot randomly access a node in a tree.
- For **generic trees** are two ways which we use to traverse a tree:
  - **Preorder Traversal**
  - **Postorder Traversal**

12

Tree Traversals – Preorder Traversal

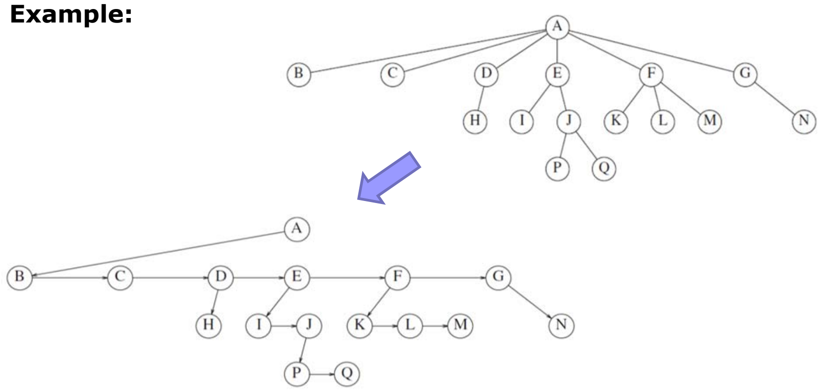
- In a **preorder traversal**, work at a parent is performed **before** (pre) its children are processed.
- **Preorder***:* root, most left-subtree, …, most right-subtree
- **Example:**



- **Preorder traversal:** A, B, C, D, H, E, I, J, P, Q, F, K, L, M, G, N
- **The preorder traversal is O(*N*)**

Tree Traversals – Postorder Traversal

- In a **postorder traversal**, work at a node is performed **after** (post) its children are evaluated.
- **Postorder:** most left-subtree, …, most right-subtree, root
- **Example:**



- **Postorder traversal**: B, C, H, D, I, P, Q, J, E, K, L, M, F, N, G, A
- **The postorder traversal is O(*N*)**

14