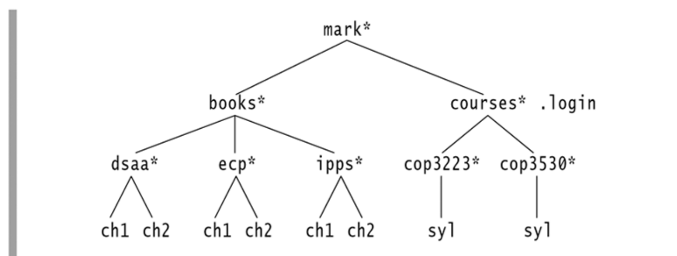
Trees

Introduction and terminology

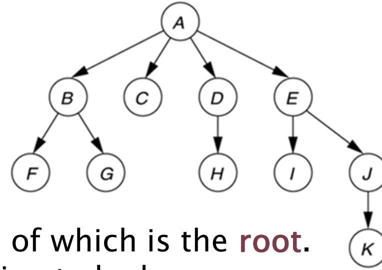
Trees in everyday (geek) life

- Class hierarchy tree (single inheritance only)
- Directory tree in a file system
  - Soon we will look at code for examining directories and their files.

figure 18.4
A Unix directory



What is a (general) tree?

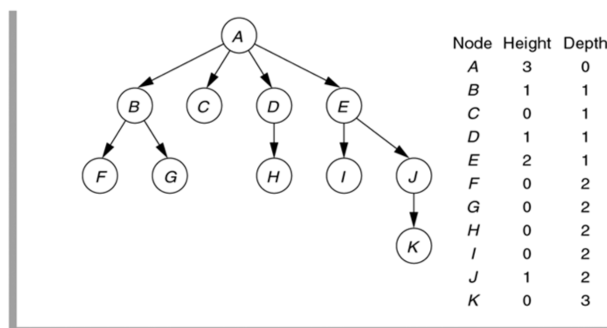


- Global view:
  - A collection of nodes, one of which is the root.
  - Nodes are connected by directed edges.
  - The root has no incoming edges.
  - Each other node has exactly one incoming edge.
  - For each node, there is exactly one path from the root to that node.
  - Usually drawn upside-down or sideways
  - **Terminology**: parent, child, grandparent, sibling, ancestor, descendent, proper ancestor, proper descendents, subtree, leaf, interior node, depth of a node, height of a node, height of the tree.

Node height and depth examples

- Which do you think is larger, the sum of the heights or the sum of the depths of all nodes in a tree?

figure 18.1
A tree, with height and depth information



Node	Height	Depth
A	3	0
B	1	1
C	0	1
D	1	1
E	2	1
F	0	2
G	0	2
H	0	2
I	0	2
J	1	2
K	0	3

What is a (general) tree?

- Recursive view:
  - A root.
  - Zero or more disjoint subtrees (none of the subtrees contain the root), each of which is a tree

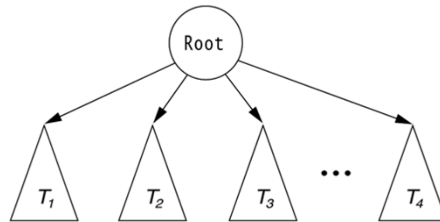


figure 18.2
A tree viewed recursively

Implementing a general tree

- Approach 1:
  - A node with N children contains N links, one for each child
  - The problem how many links should a node actually have?
  - Wasted space!
  - Solution: array of nodes
- Approach 2:
  - Each node contains a link to its first child (If any), and
  - A link to its next sibling (if any)

A Tree and its "Approach 2" implementation

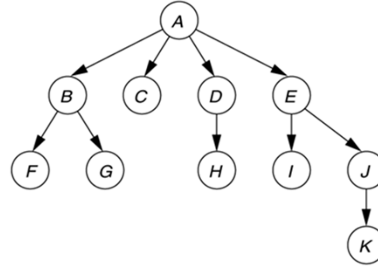
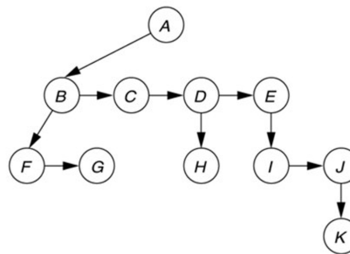


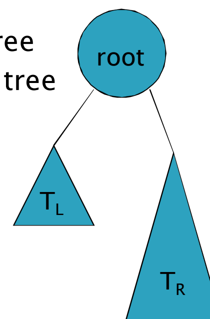
figure 18.3

First child/next sibling representation of the tree in Figure 18.1



Binary Tree: Recursive definition

- A Binary Tree is either
  - empty, or
  - consists of:
    - a distinguished node called the root, which contains an element, and
    - A left subtree T_L, which is a binary tree
    - A right subtree T_R, which is a binary tree



Binary Tree implementation

```
 1 // BinaryNode class; stores a node in a tree.
 2 //
 3 // CONSTRUCTION: with no parameters, or an Object,
 4 //     left child, and right child.
 5 //
 6 // *******************PUBLIC OPERATIONS**********************
 7 // int size( )            --> Return size of subtree at node
 8 // int height( )          --> Return height of subtree at node
 9 // void printPostOrder( ) --> Print a postorder tree traversal
10 // void printInOrder( )   --> Print an inorder tree traversal
11 // void printPreOrder( )  --> Print a preorder tree traversal
12 // BinaryNode duplicate( )--> Return a duplicate tree
13
14 class BinaryNode<AnyType>
15 {
16     public BinaryNode( )
17       { this( null, null, null ); }
18     public BinaryNode( AnyType theElement,
19                        BinaryNode<AnyType> lt, BinaryNode<AnyType>
20       { element = theElement; left = lt; right = rt; }
21
22     public AnyType getElement( )
23       { return element; }
24     public BinaryNode<AnyType> getLeft( )
25       { return left; }
26     public BinaryNode<AnyType> getRight( )
27       { return right; }
28     public void setElement( AnyType x )
29       { element = x; }
30     public void setLeft( BinaryNode<AnyType> t )
31       { left = t; }
32     public void setRight( BinaryNode<AnyType> t )
33       { right = t; }
34
35     public static <AnyType> int size( BinaryNode<AnyType> t )
36       { /* Figure 18.19 */ }
37     public static <AnyType> int height( BinaryNode<AnyType> t )
38       { /* Figure 18.21 */ }
39     public BinaryNode<AnyType> duplicate( )
40       { /* Figure 18.17 */ }
41
42     public void printPreOrder( )
43       { /* Figure 18.22 */ }
44     public void printPostOrder( )
45       { /* Figure 18.22 */ }
46     public void printInOrder( )
47       { /* Figure 18.22 */ }
48
49     private AnyType             element;
50     private BinaryNode<AnyType> left;
51     private BinaryNode<AnyType> right;
52 }
```

Build up a particular tree by calling the constructor.

size, height for tree, node

duplicate

contains

```
 1 // BinaryTree class; stores a binary tree.
 2 //
 3 // CONSTRUCTION: with (a) no parameters or (b) an object to
 4 //    be placed in the root of a one-element tree.
 5 //
 6 // *******************PUBLIC OPERATIONS**********************
 7 // Various tree traversals, size, height, isEmpty, makeEmpty.
 8 // Also, the following tricky method:
 9 // void merge( Object root, BinaryTree t1, BinaryTree t2 )
10 //                          --> Construct a new tree
11 // *******************ERRORS*********************************
12 // Error message printed for illegal merges.
13
14 public class BinaryTree<AnyType>
15 {
16     public BinaryTree( )
17       { root = null; }
18     public BinaryTree( AnyType rootItem )
19       { root = new BinaryNode<AnyType>( rootItem, null, null ); }
20
21     public BinaryNode<AnyType> getRoot( )
22       { return root; }
23     public int size( )
24       { return BinaryNode.size( root ); }
25     public int height( )
26       { return BinaryNode.height( root ); }
27
28     public void printPreOrder( )
29       { if( root != null ) root.printPreOrder( ); }
30     public void printInOrder( )
31       { if( root != null ) root.printInOrder( ); }
32     public void printPostOrder( )
33       { if( root != null ) root.printPostOrder( ); }
34
35     public void makeEmpty( )
36       { root = null; }
37     public boolean isEmpty( )
38       { return root == null; }
39
40     public void merge( AnyType rootItem,
41                        BinaryTree<AnyType> t1, BinaryTree<AnyType> t2 )
42       { /* Figure 18.16 */ }
43
44     private BinaryNode<AnyType> root;
45 }
```

Note to students

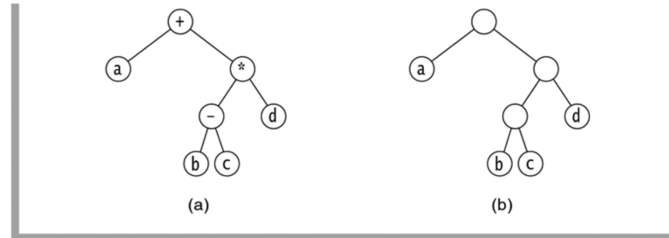
- In Spring, 2007-2008, we did not get beyond this point on Day 9.
- I left the rest of the slide in, in case you want to look ahead (in particular, to do the last couple of problems on WA6).

Binary tree traversals

- Preorder (top-down)
- PostOrder (bottom-up)
- InOrder
- LevelOrder

figure 18.11

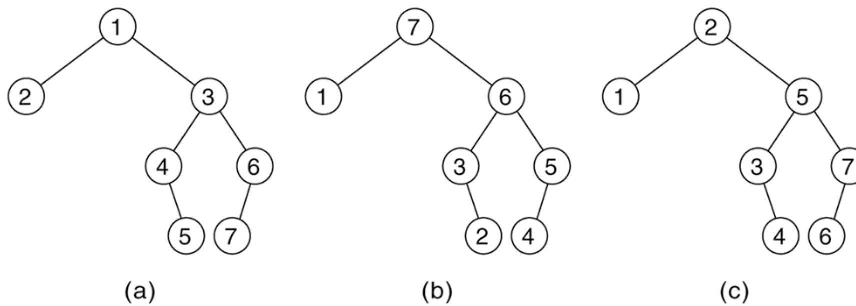
Uses of binary trees: (a) an expression tree and (b) a Huffman coding tree



Tree Traversal orders

Figure 18.23

(a) Preorder, (b) postorder, and (c) inorder visitation routes



Data Structures & Problem Solving using JAVA/2E Mark Allen Weiss © 2002 Addison Wesley

```java
// Print tree rooted at current node using preorder
public void printPreOrder( )  {
    System.out.println( element );        // Node
    if( left != null )
        left.printPreOrder( );            // Left
    if( right != null )
        right.printPreOrder( );           // Right
}

// Print tree rooted at current node using postorder
public void printPostOrder( ) {
    if( left != null )
        left.printPostOrder( );           // Left
    if( right != null )
        right.printPostOrder( );          // Right
    System.out.println( element );        // Node
}

// Print tree rooted at current node using inorder t
public void printInOrder( ) {
    if( left != null )
        left.printInOrder( );             // Left
    System.out.println( element );        // Node
    if( right != null )
        right.printInOrder( );            // Right
```

Binary Search Trees

- A Binary Search Tree (BST) is a Binary tree with the following additional properties.
  - The elements are Comparable.
  - No duplicates are allowed.
  - If the tree T is non-empty , all elements in T's left subtree are less than the root element.
  - if the tree T is non-empty, all elements in T's right subtree are greater than the root element.
  - Both subtrees are BSTs.