Topics: Simple trees, Tree traversals, Operator trees

1 Announcements

- Project 1 due date moved to July 14, 2002 11:59PM
- Read the newsgroup! (hints on HW2, more information about the project 1 due date being moved, and a sample .class file for HW2 4a)
- Reading: Data Structures (Into Java) Ch. 2-4, 5-5.3; Goodrich & Tamassia Ch. 4-6 – in particular, 5-5.1 in Data Structures (Into Java) will be likely be helpful for the project

2 Simple trees

- A tree is either empty or is a node possibly with non-empty children, which can be referred to by their position
- Each node may have a label and other stored information
- Each child only has one parent node
- Child nodes are siblings if they have the same parent node
- A node without children is a leaf node, other nodes are internal nodes
- Trees are recursive structures, and as such, recursion is often the easiest way of processing them
- A tree in which nodes have two children is called a binary tree
- A tree in which nodes have k children is called a k-ary tree, where k is known as the degree
- A set of trees is a forest

3 A general tree

class Tree {
    Object label;
    Vector children;

    Tree(Object label) {
        this.label = label;
        children = new Vector();
    }
}
4 A binary tree

class BinaryTree {
    Object label;
    BinaryTree left;
    BinaryTree right;

    BinaryTree(Object label, BinaryTree left, BinaryTree right) {
        this.label = label;
        this.left = left;
        this.right = right;
    }
}

5 Tree traversals

- Allow us a way of extracting information (in an ordered fashion) from a tree
- Typically divided into two classes – depth-first and breadth-first
- Depth-first traversals visit children before siblings
- Breadth-first traversals visit siblings before children
- Children are typically visited in order (left to right)

6 Depth-first traversals (on binary trees)

- Pre-order – visit the current node, then visit children recursively
- In-order – visit the left child recursively, visit the current node, then visit the right child recursively
- Post-order – visit children recursively, then visit the current node

Example

static void preorder(BinaryTree T) {
    if (T != null) {
        System.out.print(T.label);
        preorder(T.left);
        preorder(T.right);
    }
}

static void inorder(BinaryTree T) {
    if (T != null) {
        inorder(T.left);
        System.out.print(T.label);
        inorder(T.right);
    }
}

static void postorder(BinaryTree T) {
if (T != null) {
    postorder(T.left);
    postorder(T.right);
    System.out.print(T.label);
}

7 Question

Trees are not allowed to have any circularity (that is, there should be no path in the tree such that you can
return to a node you started at). Given a BinaryTree T, how would you test for circularity (only including
the given node, not the entire tree structure)?

White: Perform a preorder traversal on the tree and see what output you get.

Blue: Perform a postorder traversal on the tree and see what output you get.

Yellow: Modify any traversal to include the starting node T as an argument, perform the traversal, and
check whether any children are equal to the starting node.

Green: Modify any traversal to add every node visited (printed) to a Vector, and check whether the Vector
ever has more than the total number of nodes in the tree.

Pink: Huh? Wha?

Green is the answer in this case – we could determine this by elimination, since white and blue will
perform traversals forever, pink is a non-answer, and green makes assumptions about the structure of the
tree (and so will potentially loop through the same visited nodes many times before detecting circularity,
and may detect circularity for nodes other than the current one inadvertently).

8 Operator trees

- Can represent a simple expression with values and operators as a tree
- Each node is either a binary operator, in which case it has two children, or it is a value, in which case
  it has no children
- Can recursively evaluate the expression using overridden methods

Example

abstract class OpTree {
    abstract int eval();
}

class ValTree extends OpTree {
    int value;

    int eval() {
        return value;
    }
class PlusTree extends OpTree {
    OpTree left, right;

    int eval() {
        return left.eval() + right.eval();
    }
}

class MulTree extends OpTree {
    OpTree left, right;

    int eval() {
        return left.eval() * right.eval();
    }
}