STUDENT OUTLINE

Lesson 34 – Binary Trees

INTRODUCTION: A binary tree is a different kind of data structure that demands new terminology and algorithms. A binary tree node will have two pointers available for linking with other nodes, resulting in diagrams that look like inverted trees. A binary tree will begin with one node at the top and branch out below. As you might expect, the potential of going one of two different ways leads to some challenging programming problems. In fact, the idea of trying one direction, then backtracking must lead to recursion. The recursive algorithms used to traverse binary trees are very elegant and compact, but difficult to understand. But, one step at a time; first we need to learn how to talk about binary trees (vocabulary) and then build one (an algorithm).

The key topics for this lesson are:

A. Binary Tree Vocabulary
B. Building a Binary Tree
C. Shape of a Binary Tree

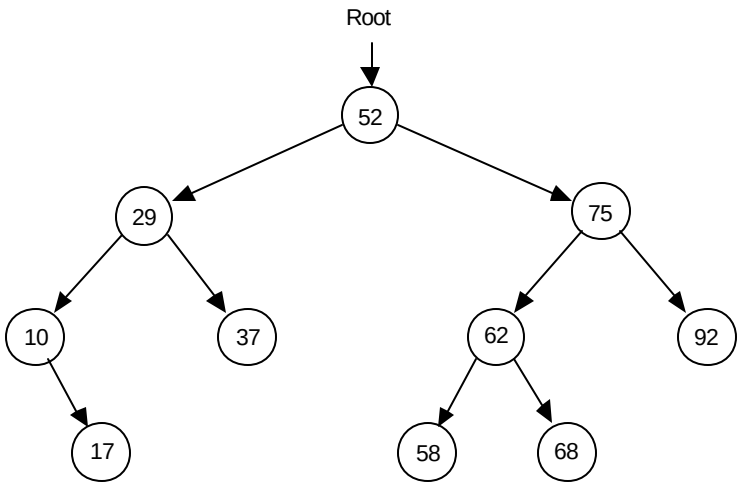
VOCABULARY:

BINARY TREE	ROOT NODE
PARENT NODE	CHILD NODE
LEAF	SUBTREE
LEVELS	EDGE

DISCUSSION:

A. Binary Tree Vocabulary

1. A binary tree is a data structure where each node has two pointers, each pointing to another node or a **`null`** value.



2. The following binary tree terms will be defined and applied to the above example.

   a. Root node - the top node in the tree; the node whose value is 52.

   b. Parent node - a node which points to one or two nodes.

   c. Child node - the node being pointed to from a parent; every node in the tree is a child to another node, except for the root node.

   d. Leaf - a node that has no children

   e. Level - the distance from the root, calculated by counting the shortest distance from the root to that node. Examples: 29 is the value stored in a node at level 1, 62 is a value stored in a node at level 2, 17 is the value of a node stored at level 3, etc.

   f. Edge - an edge joins two nodes. In the above diagram each arrow represents an edge.

3. This tree is an example of an ordered binary tree that has the following property. For every parent node, a child to the right will have a larger value, while a child to the left will have a smaller value.

4. A subtree is the entire left branch or right branch of a node. For example, the left subtree of the node containing 52 has 4 nodes. The right subtree of node containing 75 has only 1 node.

5. A leaf will have two **`null`** pointers.

B. Building a Binary Tree

1. The following definitions will apply in this next section on building a binary tree.

```
public class TreeNode
{
  private Object value;
  private TreeNode left;
  private TreeNode right;

  public TreeNode(Object initValue, TreeNode initLeft,
                  TreeNode initRight)
  {
    value = initValue;
    left = initLeft;
    right = initRight;
  }
```

```
  public Object getValue()
  {
    return value;
  }

  public TreeNode getLeft()
  {
    return left;
  }

  public TreeNode getRight()
  {
    return right;
  }

  public void setValue(Object theNewValue)
  {
    value = theNewValue;
  }

  public void setLeft(TreeNode theNewLeft)
  {
    left = theNewLeft;
  }

  public void setRight(TreeNode theNewRight)
  {
    right = theNewRight;
  }
}
```

2. Suppose the following integers were inserted into a binary tree in this order:

   26 79 14 99 53 9 35 21 87

   Draw the resulting binary tree:

See Transparency T.A.34.1, *Building a Binary Tree,* for assistance.

3. You will notice that new information was placed in the binary tree as a leaf. The insert algorithm will be a recursive solution.

4. Given this parameter list for the `insert` method, develop the pseudocode below it.

```
void insert (TreeNode node, Object data)
// Will insert data into an ordered binary tree.
// The solution is recursive.
```

C. Shape of a Binary Tree

1. The shape of a binary tree will affect its performance as a data structure and is dependent on the order of the data set.

2. If the data set came in as a sorted list (1 2 3 4 ...), the binary tree is essentially a linked list with an unused left pointer in each node.

3. A data set in random order will give a more balanced tree.

4. Ideally, we want binary trees that are balanced with almost equal numbers of nodes in each subtree of a node. The characteristic of a balanced binary tree is defined as follows: for every node in the tree, the number of nodes in its left subtree is equal to the number of nodes in its right subtree, plus or minus one. Balancing binary trees will not be covered in this curriculum guide.

SUMMARY/ REVIEW: You have learned how to build a binary tree; but is it correct? We need to "see" the tree by printing it out in ascending order. Examine the binary trees developed in this lesson and think about the task of printing out the values in order. The next lesson will introduce you to a recursive solution to this problem, as well as a few

other binary tree algorithms.

ASSIGNMENT: Lab Exercise L.A.34.1, *BSTree*

LAB EXERCISE

BSTree

Background:

In previous lessons you stored a data file, (*file20.txt*), in different data structures: an array of structures, linked list, and doubly-linked list. In the linked-list lab, we built the data structure, printed it out, searched for values, and deleted values. We will solve the same fundamental tasks using a binary tree as the data structure. You can build the binary tree in this first session, but the rest of the algorithms will be left as program stubs.

Instructions:

1. You may assume the following type definitions apply in this lab exercise:

```
public class TreeNode
{
  private Object value;
  private TreeNode left;
  private TreeNode right;

  public TreeNode(Object initValue, TreeNode initLeft, TreeNode initRight)
    { value = initValue; left = initLeft; right = initRight; }
  public Object getValue() { return value; }
  public TreeNode getLeft() { return left; }
  public TreeNode getRight() { return right; }
  public void setValue(Object theNewValue) { value = theNewValue; }
  public void setLeft(TreeNode theNewLeft) { left = theNewLeft; }
  public void setRight(TreeNode theNewRight) { right = theNewRight; }
}
```

2. Build a main menu with the following choices:

   (1) Read a file from disk, build the binary tree
   (2) Print the tree in order
   (3) Search the tree
   (4) Delete from the tree
   (5) Count the nodes in the tree

3. Start with the source code for the ordered linked-list lab from Lesson 31. The `readData` method will require small changes. The algorithms inside of all the routines will change, but the basic structure of the program will not.

4. Complete the code for the `insert` method. A precondition of the insert routine; the data file will contain no duplicate id values.

5. Stub the rest of the menu choices.

6. You may compile and run this program, but you cannot verify if your insert algorithm worked until you learn the material in Lesson 35. Use a data file as provided by your instructor.

7. Turn in one final assignment when all the menu choices are completed in Lesson 36.