Activity #7: Binary Trees

Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with ) below.

(a) Model 1, Question #7

(b) Model 1, Question #18

(c) Model 1, Question #25

(d) Model 2, Question #27

(e) Model 2, Question #38

(f) Model 2, Question #39

Activity #7: Binary Trees

A tree is a fundamental data structure in computing.

Content Learning Objectives

After completing this activity, students should be able to:

→ Explain a tree

→ Explain the properties of a tree

Process Skill Goals

During the activity, students should make progress toward:

→ Write code that traverses a tree

Model 1 Trees

Below is an hierarchical filesystem storing a collection of files. Files marked by arrows are directories. Files marked by bullets are data files.

- → /home
  - → /Documents
    - ◦ report.txt
    - ◦ source.txt
    - ◦ lab_3.txt
  - → /Desktop
    - ◦ midterm.doc
    - ◦ contacts
    - ◦ model4
  - → /Pictures
    - ◦ 332450.png
    - ◦ profile.jpg
    - → /vacation2012

Figure 1: Filesystem

1. How many directories are there?

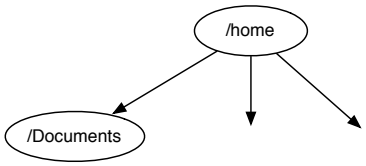
2. How many data files are there?

3. Imagine this filesystem being represented visually as it would be on your computer: as a series of folders, windows, and files. How would you navigate from the `/home` directory to `model4`?

4. Are there any files which can not be reached from the `/home` directory ?

5. Using only Figure 1, is the `/home` directory located inside of any other directories?

6. Illustrate the filesystem beginning with the `/home` directory provided below. For files, draw a node (midterm.doc). To illustrate that a file is located within another, draw a directed edge (⟶).



7. Compare your illustration to the structure below. Does the orientation of the information affect how you interpret the information being conveyed?

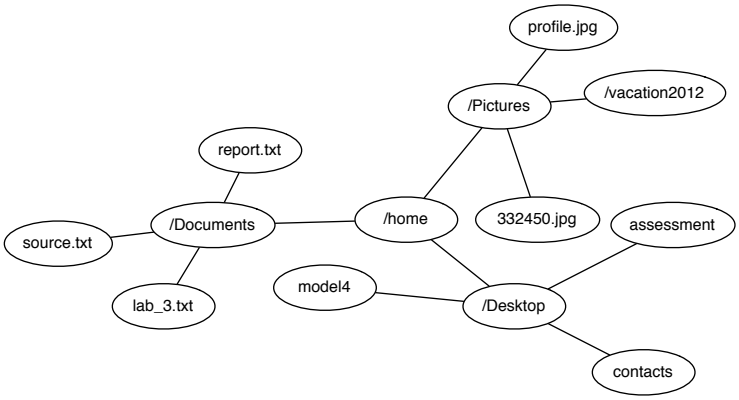


Figure 2: Alternative structure

8. Is it clear in Figure 2 that the `/home` node is in any way distinguished?

9. In your illustration, the `/home` directory node is the ***root*** of the structure. Define what it means for a node to be the root.

10. The graph you drew is an illustration of a ***tree data structure***. Using the terms "node," "edge," and "root," define tree data structure.

11. `report.txt` is a ***child*** node of the `/Documents` node. The `/Documents` node, in turn, is the ***parent*** of the `report.txt` node. What does it mean for a node to be a child? A parent?

12. Are there any nodes in the tree that do not have a parent? How many?

13. Are there any nodes in the tree that do not have any children? How many?

14. `report.txt`, `model4`, and `/vacation2012` are examples of ***leaf*** nodes. `/home` and `/Pictures` are not. Define what it means for a node to be a leaf.

15. Starting from the `/home` node how many edges are traversed navigating to the `contacts` node?

16. The `contacts` node is at a ***depth*** of 2. Define the depth of a node.

17. How deep is the tree?

18. If a tree has 16 nodes, what is the maximum possible depth of the tree? The minimum?

19. How many children does `/home` have? `/Documents`? `/vacation2012`?

20. The tree that you drew has a ***branching factor*** of 3. Define branching factor.

21. What is the branching factor of the tree below?

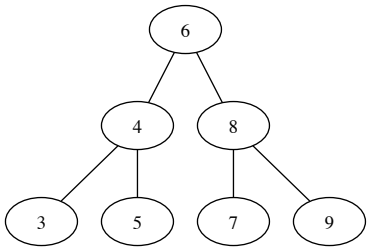


Figure 3:

22. The tree in Figure 3 is a ***binary*** tree. Define binary tree.

23. If a binary tree has 16 nodes, what is the maximum possible depth of the tree? The minimum?

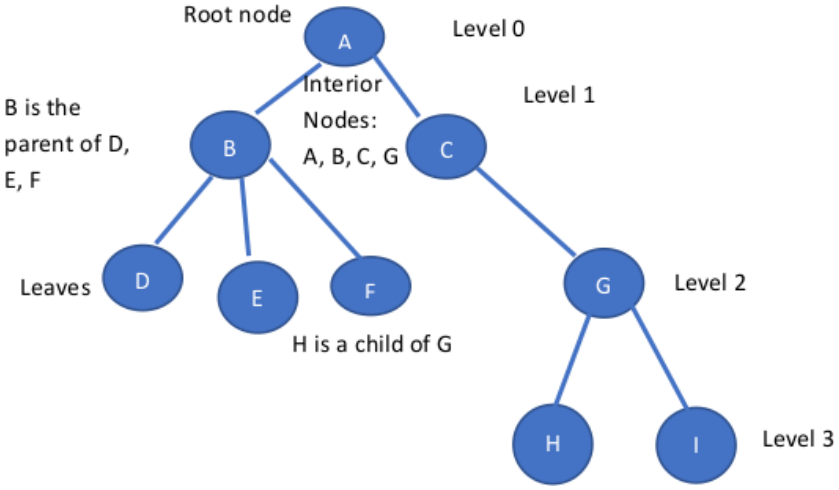
24. Illustrate the management structure within an organization with the following employees:

President
Director of IT
Director of Customer Service (CS)
Director of Sales
IT Manager
CS Supervisor
Sales Coordinator
Telephone Support
IT Specialist
Salesman

25. What data structure did you choose to use? What properties of the data structure motivated that choice?

Model 2 Binary Trees

A tree is a fundamental data structure in computing. We generally draw trees with the root node at the top of the tree (upside-down from regular trees you see in nature).



Some more vocabulary about trees:
Children are ordered left to right; a parent could have 0 or more children.
A tree with 0 nodes is an *empty* tree.
Ancestors of a node N are the nodes in the path from N to the root of the tree.
Descendants of a node N are the set of nodes that can be reached from any downward path from N.
The *height* of the tree is the number of nodes along the longest (deepest) path of the tree. The height of an empty tree is 0.
The *subtree* at node N is node N with all its descendants.

26. How could trees be useful for modeling data in computing applications?

→ Example: From Java, class hierarchies.

→ Example: From Coding, recursive function calls (hw 2, mazes).

→ Example: From Data Compression, Huffman coding.

→ Example: From Phone Menus, sequence of instructions (press 1 if you want to make a reservation, press 2 if you want to check on a reservation, press 3 if you want to speak to an operator; pressing 1 then asks, press 1 if you are making a reservation within the US, press 2 if you are making a reservation within Canada, etc.)

A **BINARY** tree is a tree in which each node has at most two children. Children are named left child or right child.
Each node contains:

Data (key,value) pairs if a dictionary
Left child
Right child

```
typedef int TreeData; // can change type with primitive or struct type
typedef struct TreeNodeTag TreeNode;

struct TreeNodeTag {
  TreeData value; // value stored in node
  TreeNode * left; // left child
  TreeNode * right; // right child
};
```

The left child is a pointer to another tree node. The right child is a pointer to another tree node. For simplicity, we will store just one data item per node (but this data item could be a struct that contains key, value pairs).

Suppose T is a binary tree, where each node has at most two children.

27. Draw a binary tree with 6 nodes (labeled A, B, C, D, E, F) where there are exactly 3 leaves.

Recall that the height of a tree is the number of nodes of the longest (deepest) path from the root to a leaf node.

What is the height of your tree?

28. Draw a binary tree with 6 nodes where there is exactly 1 leaf.

What is the height of your tree?

Assume a binary tree has height H.

29. What is the maximum # of leaf nodes in a tree of height H?

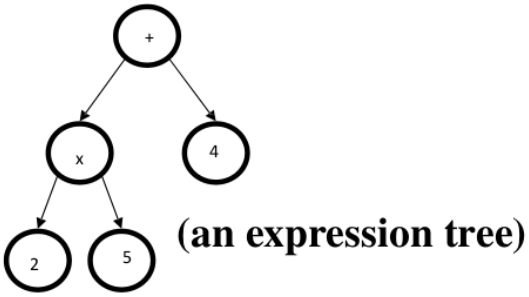
30. What is the maximum # of nodes in a tree of height H? (pack them in)

31. What is the minimum # of leaf nodes in a tree of height H?

32. What is the minimum # of nodes in a tree of height H?

In general, trees only speed things up if the tree is "full", meaning that we have close to the maximum number of nodes in a tree for a given height. A long, skinny tree does not outperform a linked list.

Here is a binary tree of arithmetic expressions.



(an expression tree)

We can traverse the tree in one of three ways:

→ Preorder: examine node, left subtree, right subtree

→ Inorder: examine left subtree, node, right subtree

→ Postorder: examine left subtree, right subtree, node

If you get confused about the names, think about ***when*** the node is examined. First (pre), Second (in), Third (post).
In the example above, here are the orders of these traversals:

→ Preorder: + x 2 5 4

→ Inorder: 2 x 5 + 4

→ Postorder: 2 5 x 4 +

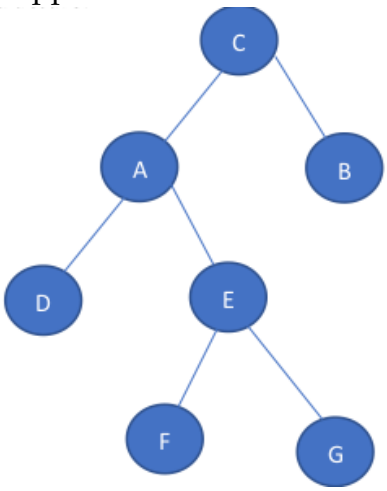
Sometimes, the order of traversal does not matter for certain operations. For example, if you want to know how many nodes are in a tree, the way you traverse the tree does not impact your result. Any traversal would be fine. Sometimes, though, order does matter. If you want to print a tree such that each level of a tree is indented further to the right, you would want to examine the tree in preorder fashion. If you are evaluating an expression tree, such as the one above, you would want to do this in postorder (get value of children before processing new operator). Here is the code for inorder traversal. Note that visit is also defined for visiting the node.

```
/* inorder
 * visits the nodes inorder (left, current, right) traversal
 */
void inorder(TreeNode * t) {
  if(t != NULL) {
    inorder(t->left);
    visit(t);
    inorder(t->right);
  }
}
```

33. Write the code to do preorder traversal:

```
void preorder(TreeNode * t) {









}
```

Suppose a tree has this structure:



34. What is the inorder traversal of this tree?

35. What is the preorder traversal of this tree?

36. Write the recursive function to return the number of leaves in a tree. This should be written recursively, since the tree data structure is recursive. Recall that if t is null, there are no leaves. If it's right child and left child are both null, t is a leaf, so return 1. Else, add together the recursive calls to process the left subtree and right subtree.

```
int numLeaves(TreeNode * t) {




}
```

37. Write a function to count the number of interior nodes with two children.

38. Assume a binary tree is traversed in-order and preorder. Here is the output. What does the tree look like?
Inorder: E F D B A C G H
Preorder: D E F G A B C H

39. What questions does your group have about binary trees?