Chapter 8 – 5 Points

**1 – 0.5 Points) The following questions refer to the tree of Figure 8.1.3**

* 1. Which node is the root?
     1. /user/rt/courses
  2. What are the internal nodes?
     1. /user/rt/courses
     2. CS016/
     3. Homeworks/
     4. Programs/
     5. CS252/
     6. Projects/
     7. Papers/
     8. Demos/
  3. How many descendants does node cs016/ have?
     1. 10
  4. How many ancestors does node cs016/ have?
     1. 1
  5. What are the siblings od node homeworks/
     1. Grades
     2. Programs/
  6. Which nodes are in the subtree rooted at node projects/?
     1. Projects/
     2. Papers/
     3. Buylow
     4. Sellhigh
     5. Demos/
     6. Market
  7. What is the depth of node papers/?
     1. 3
  8. What is the height of the tree?
     1. 4

**5 – 1 Point) Describe an algorithm, relying only on BinaryTree Operations, that counts the number of leaves in a binary tree that are the left child of their respective parent.**

Function int leftCheck(Node root):

Int count = 0

If(root.left == null AND root.right == null)

Return 0;

If root.left is not null AND root.left.left is null AND root.left.right is null

Count = 1;

If root.left is not null

Count = count + leftCheck(root.left)

/\* We could also adjust this so that this function doesn’t

Happen if the above is true because that would mean

That inherently a left lead doesn’t have further left

Nodes to check. But for simplicity, I did not implement\*/

If root.right is not null

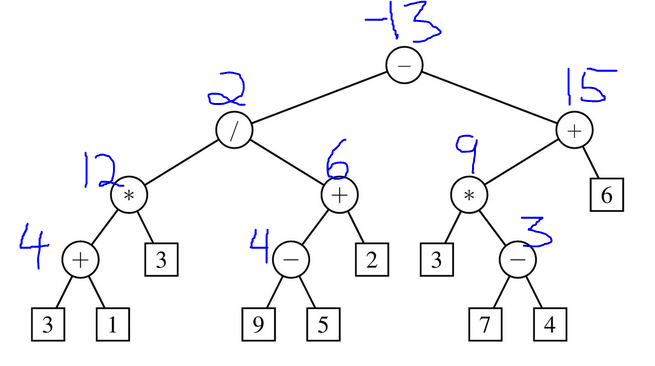
Count = count + leftCheck(root.right)

Return count

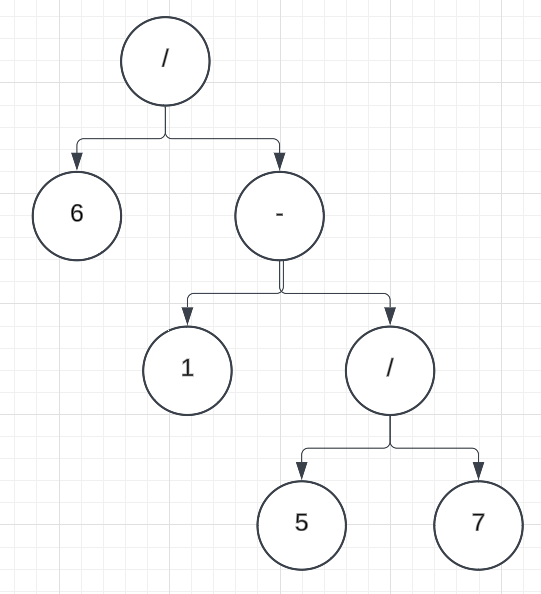
This algorithm should recursively call down the tree and add 1 to the total every time there is a left leaf node. It will then add up all the left leaf nodes that were under both the right and left node up once it gets back to the first call.

**10, 11, 12 – 0.5 Points)**

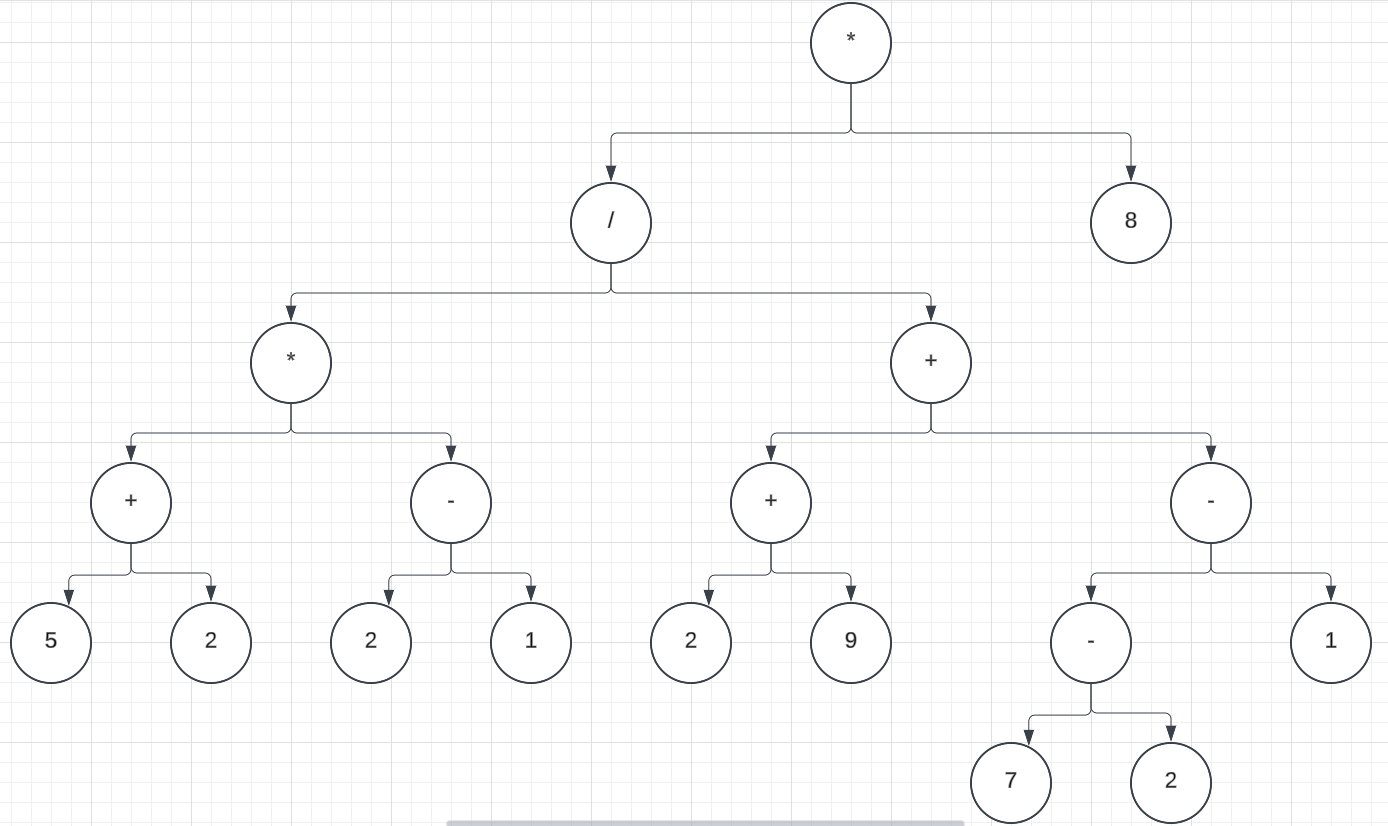
**10) Find the value of the arithmetic expression associated with each subtree of the binary tree of figure 8.2.2**



**11) Draw an arithmetic expression tree that has four external nodes, storing the numbers 1, 5, 6, and 7 (with each number stored in a distinct external node, but not necessarily in this order), and has three internal nodes, each storing an operator from the set {+, -, x, /}, so that the value of the root is 21. The operators may return and act on fractions, and an operator may be used more than once.**

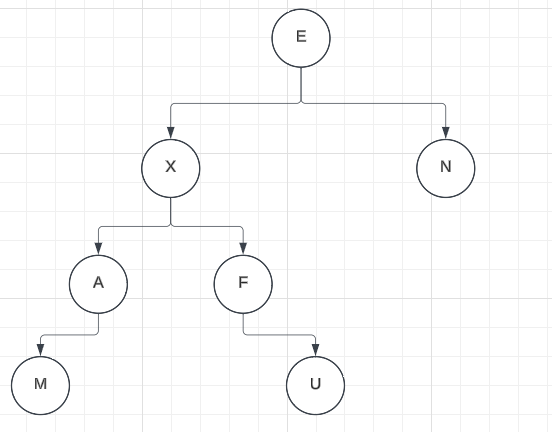


**12) Draw the binary tree representation of the following arithmetic expression:  
(((5 + 2) \* (2 - 1))/((2 + 9) + ((7 - 2) - 1)) \* 8)**



**22 – 0.5 Points) Draw a binary tree T that simultaneously satisfies the following:**

* Each internal node of T stores a single character
* A preorder traversal of T yields EXAMFUN
* An inorder traversal of T yields MAFXUEN



**28 – 0.5 Points) The path length of a tree T is the sum of the depths of all positions in T. Describe a linear-time method for computing the path length of a tree T.**

function pathCheck(Node root, int n) // N should initialize as 0

int count = n;

if root.left is not null

count = count + pathCheck(node.left, n+1)

if root.right is not null

count = count + pathCheck(node.right, n+1)

return count;

This algorithm will check each node and add its current depth to the overall path length of the function. It checks if left and right child exists and recursively calls itself accordingly. If neither left or right exists, it merely returns the current n value and then goes back up the call stack. It also does so in linear time by only checking each node once.

**41 – 1 Point) Give an efficient algorithm that computes and prints, for every position p of a tree T, the element of p followed by the height of p’s subtree.**

Function int printHeightValue(Node root)

int heightLeft = 0;

int heightRight = 0;

if root.left is not null

heightLeft = 1 + printHeightValue(root.left)

if root.right is not null

heightRight = 1 + printHeightValue(root.right)

int maxHeight = Max(heightLeft, heightRight)

print the value in the current node and maxHeight

return maxHeight

This recursively calls down to the bottom of the tree until there are no child nodes. At this point, it prints out the value in the node and a height of 0. As it calls upwards, it adds 1 to the child node’s height and prints the max height retrieved from left and right along with the value.

This computes in O(n) time as it only calls each node once.

**42 – 1 Point) Give an O(n) time algorithm for computing the depths of all positions of a tree T, where n is the number of nodes of T.**

function int getTotalDepth(Node root, int n) // n should be initialized at 0

int depthTotal = n

if root.left is not null

depthTotal = depthTotal + getTotalDepth(root.left, n + 1)

if root.right is not null

depthTotal = depthTotal + getTotalDepth(root.right, n + 1)

return depthTotal

This will recursively call down to the lowest position in the tree. Each recursive call will have an n value incremented by 1 so the depth increases by 1 each time the node goes down one node depth. As it recursively calls back up, the depth total returned will have both the current node’s depth and the depth total calculated and returned from both the left and right child node.