Assignment 7 – Trees

Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail. For the questions asking for justification, please provide a detailed mathematically oriented discussion. A proof is not required.

1. How many ancestors does a node at level n in a binary tree have? Provide justification.

A node in a binary tree at level n has n ancestors. Take the tree below as an example.

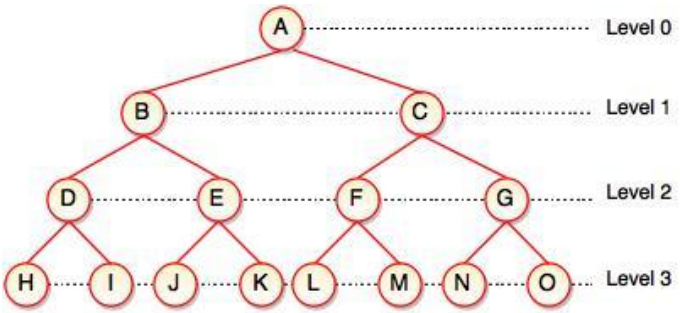

Fig. Complete Binary Tree

Take a look at node H. Node H is on level 3. Node H has ancestors D, B, and A. Therefore, node H has 3 ancestors.
Another example is node A. Node A is on level 0. This also works for the root node, because node A has 0 ancestors.

2. Prove that a strictly binary tree (regular binary tree) with n leaves contains 2n-1 nodes. Provide justification.

This assumes that the tree is full. For each leaf in the tree, two leaves have 1 parent. For every 2 parents, there is another parent. Therefore, you can represent the number of parents in a tree by:

$$\#leaves = n$$
$$\#nodes = n + \left(\frac{n}{2}\right) + \left(\frac{n}{4}\right) + \left(\frac{n}{8}\right) + \ ...$$

This series would converge to 2n. But, since there is only one root node, and we are working with integers, the solution is actually 2n-1.

3. Explain in detail that if m pointer fields are set aside in each node of a general m-ary tree to point to a maximum of m child nodes, and if the number of nodes in the tree is n, the number of null child pointer fields is n*(m-1)+1.

Each node in the tree necessarily has m child nodes and one parent node (except the first node, which has no parent node, we will get back to that later). Temporarily ignoring the first node, this means that each node that is added to the graph will add (m-1) null pointers to the graph, since we are adding m null pointers and removing 1 null pointer which points to the new node. If we have n nodes, that comes out to n*(m-1) null pointers.

But we have to remember the root node, which does not take up a null pointer. Since it does not consume a null pointer, so we add one to our count. This gives us:

$$\#nullpointers = n * (m - 1) + 1$$

4. Implement maketree, setleft, and setright for right in-threaded binary trees using the sequential array representation.

```
Class Node() {
   int index;
   int right;
   int left;

   Node(int index) {
      this.index = index;
      this.left = 2 * index;
      this.right = 2 * index + 1;
   }
}

public static setLeft(Item item) {
   Node here = currentLocation;
   if(here == null) {
      // handle error
   }
   else if(here.left != null) {
      // handle left already exists
   } else {
      Node left = Node(item)
      here.left = left;
   }
}

public static setRight(Item item) {
   Node here = currentLocation;
   if(here == null) {
      // handle error
   }
   else if(here.right != null) {
      // handle right already exists
   } else {
      Node right = Node(item)
      right.right = right;
   }
}
```

5. Implement inorder traversal for the right in-thread tree in the previous problem.

```
public item[] inorderTraversal(Tree tree) {
   Item[] orderedItem = Item[]
```

```
   here = tree.root;
   hereParent = null;
   do while(here != tree.root) { // do once, end when all nodes are added to list
      while(here.left != null) {
         hereParent = here;
         here = here.left; // go down and to the left max amount of levels
      }
      orderedItem.append(here);
      while(hereParent.right == null) {
         here = hereParent;
         hereParent = here.parent; // move up necessary amount of levels
      }
      here = here.right // traverse through tree
   }
   return orderedItem;
}
```

6. Define the Fibonacci binary tree of order n as follows: If n=0 or n=1, the tree consists of a single node. If n>1, the tree consists of a root, with the Fibonacci tree of order n-1 as the left subtree and the Fibonacci tree of order n-2 as the right subtree. Write a method that builds a Fibonacci binary tree of order n and returns a pointer to it.

```
Class FibNode() {
   public FibNode left;
   public FibNode right;
}

public FibNode fibonacciTree(int order) {
   if!(curNode) {
      pointer = FibNode; // save the value of the first node
   }
   if(order > 1) {
      curNode.left = fibonacciTree(order-1);
      curNode.right = fibonacciTree(order-2);
   }
   return pointer;
}
```

7. Answer the following questions about Fibonacci binary tree defined in the previous problem.

a) Is such a tree strictly binary?
b) What is the number of leaves in the Fibonacci tree of order n?
c) What is the depth of the Fibonacci tree of order n?

a) This tree is strictly binary. Nodes with order 2 or greater will always have 2 children, and nodes with order 1 or less will always have 0 children. Therefore the tree will always be strictly binary

b) The number of leaves in a tree of order 0 or 1 is 1. The number of leaves in a tree of order 2 is 2. The number of leaves in a tree of order 3 is 3. The number of leaves in a tree of order 4 is 5. The number of leaves in a tree of order 5 is 8. The number of leaves in the tree increases by the same increment as the previous tree + 1. These are the Fibonacci numbers.

Order	0	1	2	3	4	5
Leaves	1	1	2	3	5	6

Therefore the number of leaves is the nth fibbonacci number, as shown in the above table

c) The depth of the fibbonacci tree is equal to the order of the fibbonacci tree. The deepest depth is the left-most tree, which decreased from the value of the order to 1.