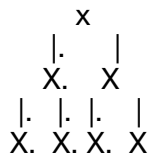
Some definitions:
- The l**evel** of a vertex is its distance from the root. The root is level 0.
- The **height** of a an m-ary tree is the highest level attained by a leaf OR length of longest path from root to a leaf.
- A **regular** m-ary (binary, tertiary) tree has 0 or m children per vertex.
- A **complete** tree of height k is regular and has all leaves at the same level.
- A **graph** is a set of vertices V and a set of edges E, comprising an ordered pair G=(V, E).

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

It has n ancestors. A node at level 0 has 0 ancestors, at level 1 has 1 ancestor (the root node), at level 2 has 2 ancestors (the parent and the root).



So the lower right node in level 2 has the one parent and the root node as ancestors. So at node at level n has n ancestors.

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

We see for any N nodes that the N + 1th level has exactly twice as many nodes.

If a tree has n leaves, it has n-1 internal nodes (probably need to prove this better, but this appears to be true)
Then, the n leaves + (n-1) internal nodes = 2n-1

It seems right:

Level	total nodes.	nodes on last level/leaves
0	1	1
1	3	2
2	7	4
3	15	8

So the total nodes = 2n-1 where n = leaves. Each level has double the last and there's always the root node to account for.

The best intuition for this is at level 2… the 4 leaves have 2 parents but those 2 parents don't each have a parent…they two parents only have 1 parent. So the 4 leaves = 2*2 - 1. The same could be said for every level.

What's a better justification? Got one?

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.

There are total number of n nodes in the tree. Each node has m pointers that can be used to point to child nodes.

M-ary trees (like binary, tertiary) have MN pointers (m pointer fields for each node). There are N-1 edges in a M-ary tree with n nodes. Therefore, the number of null child pointer fields is:
MN - (N-1) =
MN - N + 1 =
N(M-1) + 1. (distribute the N term to get to the step above)

4. **Implement maketree, setLeft, and setRight for right in-threaded binary trees using the sequential array representation**.

The key part about the sequential array representation is that parents of nodes are i/2 where i = index of the node. And children = 2i and 2i+1 where i = index of parent. So sequential array representation is:

```
class BinaryThreadedTree
        BinaryThreadedTree(int size) {
        self.Here = 0;   # this represents index of the current node
        # we have to initialize the array size off the bat. This is probably in the constructor so
that size can be variable that can be passed in.
        self.nodes = Node[] nodes = new Array[size]

}

public Node() {
        boolean rThread;
        node Right;
        node Left;
}

public Node MakeTree(String item, int index)
        # set the new item to be the element at index 0
        Node item = new Node
        nodes[index] = item

        return item

public void SetLeft(data item)
        String parent = self.nodes[self.Here]
        if parent == null
                throwError
        # cannot have data in children array indicies
        else if nodes[Here * 2] != null and nodes[Here * 2 + 1] != null
                throwError
        else
                # child on the left is self.here*2
                item = MakeTree(item, self.here*2)
                # find next open space
```

```
                until nodes[self.Here] == null
                        Here++
                # so we add the new node to the parent and the node's rThread points to
the parent since we added to the parent's left. It should point back up in an inOrder
traversal.
                item.Right = parent
                item.rThread = true;


public void SetRight(data item)
        String parent = self.nodes[self.Here]
        if parent == null
                throwError
        # cannot have data in children array indicies
        else if nodes[Here * 2] != null and nodes[Here * 2 + 1] != null
                throwError
        else
                # child on the left is self.here*2 + 1
                item = MakeTree(item, self.here*2 + 1)
                # this is the thread for the new item coming from the parent
                item.right = nodes[parent].right
                item.rThread = true
                # find next open space
                until nodes[self.Here] == null
                        Here++
                # so we add the new node to the parent and the node's rThread points to the
parent since we added to the parent's left. It should point back up in an inOrder traversal.
                nodes[parent].right = item
                nodes[parent].rThread = false
```

To grader, is this right? What else am I missing? Can you give me an example of this done well?

5. **Implement inorder traversal for the right in threaded tree in the previous problem**

```
public void traverseThreadedTree(Node root)
# passed in node should be a root
   Node current = leftmost(root);

   while (current != null)
      printf("%d ", cur->data);

      // If this node is a thread node and has a right thread then go to inorder successor
      if (current.rThreaded)
         current = current.right;
      else // Else go to the leftmost child in right subtree.If we're here, that means current is a
parent node of some kind.
         current = leftmost(current.right);
   }
```

```
}

# this gives back the most child left most node
Node leftMost(Node node)
   if (node == null)
      return Null;

   while (node.left != null)
       n = node.left;

   return n;
}
```

6. **Define the fibonacci binary tree of order n as follows: If n = 0 or n = 1 then the tree consists of a single node. If n> 1 then 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.**

A variant of a binary tree where a tree of order n (n>1) has a left subtree of order n-1 and a right subtree of order n-2. An order 0 Fibonacci tree has no nodes, and an order 1 tree has 1 node.

The root of a Fibonacci tree should contain the value of the nth Fibonacci number the left subtree should be the tree representing the computation of the n-1 Fibonacci number, and the right subtree should be the tree representing the computation of the n-2 Fibonacci number.

```
public Node fib(int n)
{
  Node node = new Node();

  if (n == 0 || n == 1) {
    node.left = null;
    node.right = null;
  }
  else {
    node.left = fib(n - 1);
    node.right = fib(n - 2);
  }

  return node; # the node represents the root node of the tree
}
```

To grader, is this right?

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

a. is such a tree strictly binary?

It seems like it? The node only has max 2 children.

b. What is the number of leaves in the Fibonacci tree of order n?

It seems like the leaves are fib(n+1).

A tree of fib(4) has fib(5) leaves — the root is 3 and it has 5 leaves. A tree of fib(5) has fib(6) leaves — 5 is the root and it has 8 leaves.

What is **order?**

c. What is the depth of the Fibonacci tree of order n?

the depth appears to be n - 1