**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.**

The answer is that there will always be n ancestors for a node at level n. This is based on definition of a level. A level of a vertex is its distance from the root. This means that we count the number of edges between the node and the root. The root starts at 0, since it has to edges leading from it to the root. The first level would have one, the second level would have two, and so on. Each edges would also connect the vertex to exactly one preceding vertex. This leads to the conclusion that number of edges = level of the node = number of ancestors.

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

A strictly binary tree means a tree with either exactly 2 children or 0 children. We can see that if it has 2 leaves, that means that it has 2\*2-1 =3 nodes, which is the root and its two children. If only one of these children has a set of twins descendant from it, we would have 3 leaves, or 3\*2-1=5 nodes. If both of these had a set of twins descendant from it, we would have 4 leaves, or 4\*2-1=7 nodes. This would hold true for any set of leaves.

This is because every time you add a leaf, you are creating two more nodes. To go from 2 to 3 leaves, a pair of nodes had to be created descendant from one of the previous nodes. In this way, the number of nodes will always be related to twice the number of leaves. The -1 is subtracted to account for the fact that we only have one node in the root, the exception to the rule that each leaf has to create two nodes.

1. **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.**

We can see that each node begins with m null child pointers. Let’s take, for example, a trinary tree. When there is only the root, there is 1 node. The equation yield 1\*(3-1)+1 =3, which is correct. When another node is added, one of the null child nodes of the root is filled. But, three more are created with the new node. This yields the equation 2\*(3-1)+1 = 5, which is correct.

In general, the root contains m null child pointers. Each additional node added creates m-1 null child pointers, because of the fact that it has to take up a null child pointer from its ancestor. Because of this, we can see the direct association of number of nodes, and m-1 null child pointers. The +1 is added because of the special case of the root, in that it did not have to take a null child pointer from an ancestor in order to exist.

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

Maybe we just put the parent indicator where the right child would be. When we set right, we compare to see if the right node is the same as the parent node. If it is, we can overwrite it with the new value. If it is different, we return an error, saying that the node already exists. We would have two checks- one to see if the field is null, and one to see if the index is within the size of the array.

//Assuming that the tree is complete, so that there will be no missing nodes before the leaves.

Class Tree {

Class Array\_Pointer{

Static int maxSize = 0;

Static String[] strArray;

Public void maketree(int levels){

Size = 2^(levels + 1); //assuming levels start at 0.

strArray = new String[size];

maxSize = size;

}

Public void set\_Left(String data, int rootIndex){

Int t = (rootIndex \* 2);

If (t>maxSize){

Return out of bounds error;

} else if (strArray[t] != null){

Return field already populated error;

} else {

strArray[t] = key;

if(2\*t+1 <= maxSize){

strArray[2\*rootIndex+1] = “{P}” //setting a pointer in the right side if there is space in the array

}

}

}

Public void set\_Right(String key, int root){

Int t = (root\*2) + 1;

If (t>maxSize){

Return out of bounds error;

} else if (strArray[t] != null || strArray[t] != “{P}”){

Return field already populated error;

} else {

strArray[t] = key;

}

}

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

//This prints the values of the array starting from the leftmost node, then its parent, then the parent’s right node.

void printInOrder(String[] strArray, int index){

if (strArray[index] == null || index > maxSize || strArray[index] == “{P}” ){

return;

}

printInOrder(index\*2);

System.out.print(strArray[index]); //prints out the value if not a pointer or null

printInOrder(index\*2+1);

}

1. **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 Fibonacci{

Class TreeNode {

DataType Data;

TreeNode Left, Right;

}

Class TreeNode(int n){

TreeNode new = new TreeNode();

If ( n == 0 || n == 1){

new.Left = null;

new.Right = null;

} else {

new.Left = tree(n-1);

new.Right = tree(n-2);

}

Return new;

}

}

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

**a) Is such a tree strictly binary?**

Yes, such a tree is strictly binary. Each node will either have no children or two children.  **b) What is the number of leaves in the Fibonacci tree of order n?**

The number of leaves in a Fibonacci tree of order n will be (the number of leaves of a tree order n-1) + (the number of leaves of a tree order n-2). Order 0 and 1 are special cases, and will always have one leaf.  **c) What is the depth of the Fibonacci tree of order n?**The depth is n, except for 0, which is 1.