Adam Rich

EN.605.202.87.SP18 Data Structures

Homework Assignment 7

March 20, 2018

**Assignment 7 – Trees**

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

The answer is n. Unlike humans, nodes in a binary tree have zero or one parents. The root node, at level 0, has 0. A child of the root node, in level 1, has 1. To complete an inductive proof one would need to show that a child node has exactly one more ancestor than its parent (if the parent exists).

Given a non-root node P at level n with ancestor set A(P) and that P has a child C. P is an ancestor of C that is not in the set A(P). We want to prove that A(C) is {A(P) union P}. Assume that there is some other ancestor of C that is not in the set {A(P) union P}, call it X. There would then be a path from C to the root through X and through A(P) which does not include X. That would imply the graph is cyclic which cannot be true for a binary tree.

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

The simplest regular binary tree available is a tree with one node (the root) and no children. It is a leaf node, there is 1, and the total number of nodes is 2(1) – 1 = 1. In any regular binary tree, a leaf node can have two children added and the tree is still regular. Both those children would be leaf nodes. So, if before it had n leaf nodes, now it has n less 1 (for the new parent) plus two (for the new children), or n + 1. The total number of nodes has increased by 2; if n increases by 1 the number of leaves increases by 2. So, it is true for n = 1. And if we assume it is true for n = k, it is also true for k + 1.

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.**

Use 3-ary complete tree as an example.

Tree Level Nodes in Level Nodes Total

r 0 1 = 3^0 1 = [3^(1+0) – 1]/(3-1)

/ | \

/ | \

a b c 1 3 = 3^1 4 = [3^(1+1) – 1]/(3-1)

/|\ /|\ /|\

d e f g h i j k l 2 9 = 3^2 13 = [3^(1+2) – 1]/(3-1)\*

The total number of nodes in a complete m-ary tree with height k is

[m^(k + 1) – 1] / (m – 1)

The total number of leaves in the same m-ary tree is

m^k

The number of NULL pointers in this tree is

m^k \* m = m^(k + 1)

= m^(k + 1) – 1 + 1

= [m^(k + 1) – 1]/(m – 1) \* (m – 1) + 1

= n \* (m – 1) + 1

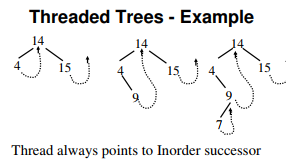
For every leaf node I remove, I remove it's m NULL pointers but add a NULL pointer for its parent. So, n goes down 1 and the number of NULL pointers goes down m – 1.

\*I looked this up...

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

From lecture: thread always points to in-order successor

in a right in-threaded tree



class TreeNode

DataType value // null if the node is not being used

int thread // the zero-indexed pos of the in-order successor

method init

value = null

int = -1 // since value is null, in-order successor is undefined

// When value is not null, -1 will mean a null successor

end-method

end-class

class Tree

private int size, predetermined constant

private array arr[] of TreeNodes, zero-indexed

private boolean isMade = false

// for zero-indexed

// children of k are 2k + 1 and 2k + 2

// parent of n is floor((n-1)/2)

method init

// populate array with null TreeNodes

for i = 0 to size-1

arr[i] = new TreeNode

end-for

isMade = false // still

end-method

method MakeTree(DataType item)

if isMade "throw error"

else

isMade = true

arr[0].value = item

arr[0].thread = -1

here = 0

phere = -1

end-method

method SetLeft(DataType item, int p) // p the parent index

// Allow overwriting

if item is null "throw error: value cannot be null"

if arr[p].value is null "throw error: parent doesn't exist"

int c = 2\*p + 1

if !isMade "throw error: create root with MakeTree"

if c >= size "throw error: array overflow"

else

arr[c].value = item

arr[c].thread = GetSuccessor(c)

return c

end-method

method SetRight(DataType item, int p) // p the parent index

if item is null "throw error: value cannot be null"

if arr[p].value is null "throw error: parent doesn't exist"

int c = 2\*p + 2

if !isMade "throw error: create root with MakeTree"

if c >= size "throw error: array overflow"

else

arr[c].value = item

arr[p].thread = c

arr[c].thread = GetSuccessor(c)

return c

end-method

private method GetSuccessor(int n)

if arr[n].value = null "throw error: unused node"

if n = 0 return null

int cr = 2\*n + 2 // right child

if arr[cr].value != null return cr

int p = floor((n-1) / 2) // parent

if n is odd (this is a left node) return p

else return GetSuccessor(p)

end-method

end-class

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

// Part of tree class, so private data available

method TraverseInorder(function F)

boolean[] done = new boolean[size]

for i = 0 to size – 1

done[i] = false

int here = 0 // start at root

int cl = 1 // left child

do while here > -1

if arr[cl].value != null AND !done[cl]

here = cl

cl = 2\*here + 1

continue // jump to next iteration

F(arr[here].value) // DO STUFF

done[here] = true

here = arr[here].thread

cl = 2\*here + 1

end-while

end-method

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 TreeNode

DataType value

TreeNode left = null

TreeNode right = null

end-class

function FibTree(int n)

if n < 0 "throw error and exit" else...

TreeNode t = new TreeNode

if n == 0 or n == 1

return t // right and left already init. to null

else

t.left = FibTree(n – 1)

t.right = FibTree(n – 2)

return t

end-function

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

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

Discussion forum clarified this is to be read "is such a tree regular"?

Regular meaning that every node has 0 or 2 children.

Yes, FibTree(0) and FibTree(1) are leaves with no children.

Given a sub-tree S = FibTree(x) x > 1, the root will have two children and those will be the root nodes of FibTree(x-1) and FibTree(x-2)

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

n Number of leaves

0 1

1 1

2 2

For 2, it is 2

2

/ \

1. 0

For 3, it is 3

3

/ \

2 1

/ \

1. 0

For 4, it would be the number of leaves on the FibTree(3) subtree plus the number of leaves on the FibTree(2) subtree, or 3 + 2 = 5. The number of leaves will \*always\* be the Fibonacci number for n.

**c) What is the depth of the Fibonacci tree of order n?**

The left-hand subtree of FibTree n is always FibTree(n – 1). So, labelling the vertices based on the n of the function call that defined them you get, for n = 5 for example

5

/ \

4 3...

/ \

3 2...

/ \

2 1

/ \

1. 0

This one has 5 levels and a height of 4. Because we are always "counting down" by one on the left had side (and something lower on the right and its children) we will always have n levels and a height of n-1.