**Conor Sweeney – cjs2201**

**COMS W3134 Spring 2016 (Sections 1 and 2)**

**Homework 3**

**Due: 4:00pm on Friday, March 4th**

**Written (35 pts)**

1. Problem 1 (6 pts): Weiss 4.6 - We're looking for a full induction proof

A full node is a node with two children. Prove that the number of full nodes plus one is equal to the number of leaves in a nonempty binary tree.

Theorem: T(N): If there are N full nodes in a nonempty binary tree there are N+1 leaves.

Basis: T(0): If there are 0 full nodes in a nonempty binary tree then there is only one leaf. This is true because it has only one branch in the tree due to the 0 full node.

Inductive Step: Show that while T(k) 🡪 T(k+1) , k >= 0

T(k+1): If there are k+1 full nodes in a nonempty binary tree then there are k+2 leaves

Pick a leaf node and remove its parents recursively until a full node is reached. The full node becomes a non full node because one of its child nodes is removed. At this point the tree will have one less leaf and one less full node.

Therefore, the tree has k full nodes after the nodes are removed and therefore, there are k+1 leaves. Add all the nodes back into the tree the same way to create the original tree. This adds one full node and one leaf node. Proving, there are k+1 full nodes with k+2 leaf nodes.

1. Problem 2 (6 pts): Weiss 4.9 - In part a show the tree after each insert - a total of 8 trees.  In part b, we're doing full deletion, not lazy deletion.

a. Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.

See images at the bottom of the document.

b. Show the result of deleting the root.

See images at the bottom of the document.

1. Problem 3 (6 pts): Show the same sequence of inserts as listed in the previous problem, except this time show the insertions into an AVL tree.  Make sure to show each rotation.

See images at the bottom of the document.

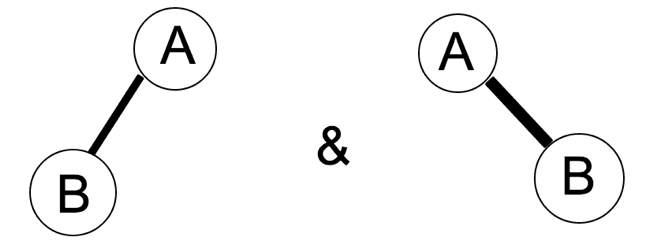
1. Problem 4 (6 pts): Prove that a preorder and postorder traversal of a binary tree is not necessarily sufficient to reconstruct the tree uniquely.

A very simple example proving this is if you are given these traversals:

Preorder: [A,B]

Postorder: [B,A]

These traversals can create two different binary trees:



Therefore, a binary tree cannot be recreated based on preorder and postorder traversals.

1. Problem 5 (6 pts): Based on Weiss 4.16 - Describe any modifications that you would need to make to the BinaryNode itself, and then show the implementation for findMin.  You don't actually have to give us a full working class.

Weiss 4.16: Redo the binary search tree class to implement lazy deletion. Note carefully that this affects all of the routines. Especially challenging are findMin and findMax, which must now be done recursively.

Add a Boolean to the BinaryNode class called deleted. Instead of removing the BinaryNode when it is lazy deleted it is now set to yes. Now the BinaryNode findMin function should be appropriately changed to reflect this. The findMin function should be changed to this:

private BinaryNode<E> findMin(BinaryNode<E> t)

    {

        BinaryNode<E> r = t;

        if (t==null)

{

            return null;

}

        //found deleted node

        if (t.deleted && t.left != null)

        {

            return findMin(t.left);

        }

        if (t.deleted && t.left == null)

        {

            if (t.right == null)

            {

                return findMin(r);

            }

            return findMin(t.right);

        }

        //found real node

        if (t.left == null && !t.deleted)

        {

            return t;

        }

        if (t.left != null && !t.deleted)

        {

            return findMin(t.left);

        }

        return null;

    }

