**LEBANESE AMERICAN UNIVERSITY**

**Department of Computer Science and Mathematics**

**CSC 310: Algorithms and Data Structures**

Fall 2013

**Lab II**

**Binary Trees**

Implement a binary tree by writing the class BTNode which represents a binary tree node. The class has an integer value and a reference to the left and right children.

Write the insert method which should insert a node on the left or the right at each level to keep the Binary tree structure and the rest of the problems as methods in the main.

**Input:**

All inputs are read from a file and the name of the file is “tree.in”. In the file, You read an integer **k** then **k** test cases follow. For each case, we read an integer **n** then **n** integer follows which specifies the number of nodes you have to insert into the tree(includes the root).

**Sample Input:**

k

**n** n1 n2 n3…

…

**Output:**

Output should be consistent with the output specified in each problem as specified.

**Problem 1**

Write a method that prints the tree level by level (BFS) starting from the left of the tree.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 25 13 30 10 15 27 37

**4** 6 7 8 9 6 7 8 9

**6** 10 7 15 13 4 6 10 7 15 4 13 6

**Problem 2**

Write a method that prints the Tree in a Depth search fashion specifically the in-order DFS.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 25 10 13 15 25 27 30 37

**4** 6 7 8 9 6 7 8 9

**6** 10 7 15 13 4 6 4 6 7 10 13 15

**Problem 3**

Write a method that counts the number of leaf nodes in each tree.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 4

**4** 6 7 8 9 1

**6** 10 7 15 13 4 6 2

**Problem 4**

Write a method that counts the number of levels in a tree. the level at the root is 0.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 2

**4** 6 7 8 9 3

**6** 10 7 15 13 4 6 3

**Problem 5**

Write a method that checks whether a tree is full or not. A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children, print to screen “**I am a full tree!**” otherwise print “**I am not a full tree!**”.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 I am a complete tree!

**4** 6 7 8 9 I am not a complete tree!

**6** 10 7 15 13 4 6 I am not a complete tree!

**Problem 6**

Write a method that counts the total number of nodes in a tree.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 7

**4** 6 7 8 9 4

**6** 10 7 15 13 4 6 6

**Problem 7**

A Binary Tree is strictly binary if every node has an even number of children (0 or 2). Write a method that checks whether the tree is Strictly Binary or not. You must print “**Strictly Binary**” for Strictly Binary trees and “**Not Strictly Binary**” for non-Strictly Binary trees.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 Strictly Binary

**4** 6 7 8 9 Not Strictly Binary

**6** 10 7 15 13 4 6 Not Strictly Binary

**Problem 8**

Write a method that searches for an element in a Binary Tree and print “**Found**” if the element was found print “**Not Found**” if the element was not found. You will read an extra element at the end which represents the extra element that is being searched.

**Sample Input: Sample Output:**

3

**7** 25 13 10 30 15 27 37 **13** Found

**4** 6 7 8 9 **8** Found

**6** 10 7 15 13 4 6 **2** Not Found

**Problem 9**

Delete the node which is found in the tree the node you want to read will be read as an extra element after you read elements in the tree, and you will print the tree as BFS. Always replace/rebranch the deleted node by its left child otherwise use the right. If it is a leaf node, delete the node.

**Sample Input: Sample Output:**

3

**7** 25 13 10 30 15 27 37 **13** 25 10 30 15 27 37

**4** 6 7 8 9 **8** 6 7 9

**6** 10 7 15 13 4 6 **4** 10 7 15 6 13

**Problem 10**

Check if a tree is AVL by checking if it has balanced heights [-1, 0, 1]. Print “**Is AVL**” if the tree is an AVL tree otherwise print “**Is Not AVL”**.

**Sample Input Sample Output**

3

**7** 25 13 10 30 15 27 37 Is AVL

**4** 6 7 8 9 Is Not AVL

**6** 10 7 15 13 4 6 Is Not AVL

**Problem 11**

You want to print the number of nodes at each level of each BST. For example, if you read a complete BST of height 3, your program prints the list 1 2 4 8 (you have the root at level 0, then two children at level 1, then four nodes at level 2 and 8 at the last level). To solve this problem you are to write the method numlevel(BST T, int L) that returns the number of nodes at level L in T. The main method calls numLevel several times (per tree) in order to get the required list (you should know how to compute the number of times numLevel is called per tree!)

**Sample Input: Sample Output**

3

**5** 12 15 4 7 8 1 2 1 1

**1** 11 1

**2** 5 6 1 1

**Problem 12**

Two binary trees **T1** and **T2** are isomorphic if and only if:

* If each of **T1** and **T2** consists of a single node (the root), then they are isomorphic.
* If **T1** and **T2** consist of more than one node, then they are isomorphic if and only if: there is a **1-1 correspondence** between the (left and right) subtrees **T1** and those of **T2** such that the corresponding subtrees are isomorphic. In other words, the right subtree of **T1** is isomorphic to the right subtree of **T2** and the left subtree of **T1** is isomorphic to the left subtree of **T2** **OR** the right subtree of **T1** is isomorphic to the left subtree of **T2** and the left subtree of **T1** is isomorphic to the right subtree of **T1**.

Write a function that checks if two tree are isomorphic which prints “**Isomorphic**” if they are else print “**Not Isomorphic**”. In each test case, you will be reading two BST trees.

**Sample Input Sample Output**

2

**5** 1 2 3 4 5

**5** 4 3 5 2 7 Not Isomorphic

**7** 5 3 2 4 7 6

**7** 10 20 30 5 3 7 17 Isomorphic

**Problem 13**

Write a method depthSum that returns the sum of the values stored in a binary tree of integers weighted by the depth of each value. You should return the value at the overallRoot plus 2 times the values stored at the next level of the tree plus 3 times the values stored at the next level of the tree plus 4 times the values stored at the next level of the tree and so on. For example, in the tree below:

**+---+  
 | 9 |  
 +---+  
 / \  
 +---+ +---+  
 | 4 | | 7 |  
 +---+ +---+  
 / \ \  
+---+ +---+ +---+  
| 2 | | 5 | | 8 |  
+---+ +---+ +---+  
 / \  
 +---+ +---+  
 | 6 | | 9 |  
 +---+ +---+**

The sum would be computed as:

1 \* 9 + 2 \* (4 + 7) + 3 \* (2 + 5 + 8) + 4 \* (6 + 9) = 27960

**Sample Input Sample Output**

1

**8** 9 4 7 5 6 2 8 9 27960

**Problem 14**

Write a method countEmpty that returns the number of empty branches in a tree. An empty tree is considered to have one empty branch (the tree itself). For non-empty trees, your method(s) should count the total number of empty branches among the nodes of the tree. A leaf node has two empty branches. A node with one non-empty child has one empty branch. A node with two non-empty children (a full branch) has no empty branches. For example the tree below has 15 empty branches (indicated by circles):

**+---+  
 | 0 |  
 +---+  
 / \  
 +---+ +---+  
 | 4 | | 6 |  
 +---o +---+  
 / / \  
 +---+ +---+ +---+  
 | 3 | | 0 | | 1 |  
 +---o +---+ o---+  
 / / \ \  
 +---+ +---+ +---+ +---+  
 | 4 | | 3 | | 8 | | 5 |  
 +---o o---+ o---o +---+  
 / \ / \  
+---+ +---+ +---+ +---+  
| 1 | | 9 | | 2 | | 7 |  
o---o o---o o---o o---o**

**Sample Input Sample Output**

1

**8** 9 4 7 5 6 2 8 9 10

**Problem 15**

Write a method countLeftNodes that returns the number of left children in the tree. A left child is a node that appears as the root of the left-hand subtree of another node. An empty tree has 0 left nodes. For example, the following tree has four left children (the nodes storing the values 5, 1, 4, and 7):

+---+  
 | 3 |  
 +---+  
 / \  
 +---+ +---+  
 | 5 | | 2 |  
 +---+ +---+  
 / / \  
+---+ +---+ +---+  
| 1 | | 4 | | 6 |  
+---+ +---+ +---+  
 /  
 +---+  
 | 7 |  
 +---+

**Sample Input Sample Output**

1

**8** 9 4 7 5 6 2 8 9 3

**Problem 16**

Write a method countEvenBranches that returns the number of branch nodes in a binary tree that contain even numbers. A branch node is one that has one or two children (i.e., not a leaf node). An empty tree has 0 even branches. For example, if a variable tree stores a reference to the following tree:

+---+  
 | 2 |  
 +---+  
 / \  
 +---+ +---+  
 | 8 | | 1 |  
 +---+ +---+  
 / / \  
+---+ +---+ +---+  
| 0 | | 7 | | 6 |  
+---+ +---+ +---+  
 / \  
 +---+ +---+  
 | 4 | | 9 |  
 +---+ +---+

Then the call to your method should return 3 because there are three branch nodes with even values (2, 8, and 6). Notice that the leaf nodes with even values are not included (the nodes storing 0 and 4).

**Sample Input Sample Output**

1

**8** 9 4 7 5 6 2 8 9 3

**Problem 17**

Write a method printLevel that accepts an integer parameter *n* and that prints the values at level *n* from the left to right, one per line. We will use the convention that the overallRoot is at level 1, that its children are at level 2, and so on. For example, if a variable tree stores a reference to the following tree:

+----+  
 | 12 |  
 +----+  
 / \  
 / \  
 +----+ +----+  
 | 19 | | 93 |  
 +----+ +----+  
 / \ \  
 / \ \  
+----+ +----+ +----+  
| 11 | | 14 | | 15 |  
+----+ +----+ +----+  
 /  
 /  
 +----+  
 | 10 |  
 +----+

Then the call printLevel(3); would produce the following output:

11  
14  
15

**Sample Input Sample Output**

1

**8** 9 4 7 5 6 2 8 9 **2** 4

7

**Problem 18**

Write a method printLeaves that outputs the leaves of a binary tree from right to left. More specifically, the leaves should be printed in the reverse order that they would be printed using any of the standard traversals. For example, if a variable tree stores a reference to the following tree:

+---+  
 | 2 |  
 +---+  
 / \  
 +---+ +---+  
 | 8 | | 1 |  
 +---+ +---+  
 / / \  
+---+ +---+ +---+  
| 0 | | 7 | | 6 |  
+---+ +---+ +---+  
 / \  
 +---+ +---+  
 | 4 | | 9 |  
 +---+ +---+

Then the call of printLeaves(); should produce the following output:

leaves: 9 4 0

If the tree does not have any leaves (an empty tree), simply print:

no leaves

**Sample Input Sample Output**

1

**8** 9 4 7 5 6 2 8 9 **2** leaves: 9 6 2

**Problem 18**

Write a method doublePositives that doubles all data values in a binary tree of integers, and prints them in BFS order. For example:

**+----+  
 | 9 |  
 +----+  
 / \  
 / \  
 +----+ +----+  
 | 3 | | 15 |  
 +----+ +----+  
 / / \  
 / / \  
+----+ +----+ +----+  
| 0 | | 12 | | 24 |  
+----+ +----+ +----+  
 / \  
 / \  
 +----+ +----+  
 | 11 | | 13 |  
 +----+ +----+**

**Sample Input Sample Output**

1

**8** 9 3 15 0 12 24 11 13 18 9 30 0 24 48 22 26

**Problem 19**

Write a method toString2 for a binary tree of integers. The method should return "empty" for an empty tree. For a leaf node, it should return the data in the node as a String. For a branch node, it should return a parenthesized String that has three elements separated by commas:

1. The data at the root.
2. A String representation of the left subtree.
3. A String representation of the right subtree.

For example, if a variable tree stores a reference to the following tree:

**+---+  
 | 4 |  
 +---+  
 / \  
 +---+ +---+  
 | 3 | | 7 |  
 +---+ +---+  
 / / \  
+---+ +---+ +---+  
| 1 | | 6 | | 9 |  
+---+ +---+ +---+  
 / \  
 +---+ +---+  
 | 5 | | 8 |  
 +---+ +---+**

Then the call tree.toString2(); should return the following String:

"(4, (3, 1, empty), (7, (6, 5, empty), (9, empty, 8)))"

**Sample Input Sample Output**

1

**8** 4 3 7 1 6 9 5 8 (4, (3, 1, empty), (7, (6, 5, empty), (9, empty, 8)))

**Problem 20**

Write a method completeToLevel that accepts an integer *n* as a parameter and that adds nodes to a tree so that the first *n* levels are complete. A level is complete if every possible node at that level is not null. We will use the convention that the overall root is at level 1, it's children are at level 2, and so on. You should preserve any existing nodes in the tree. Any new nodes added to the tree should have -1 as their data. Print the tree using BFS.

**Sample Input Sample Output**

1

**8** 4 3 7 1 6 9 5 8 **1** 4 3 7 1 -1 6 9 5 8

**Problem 21**

Given the binary Tree and the two nodes say ‘a’ and ‘b’, determine whether the two nodes are cousins of each other or not.

Two nodes are cousins of each other if they are at same level and have different parents.

6  
 / \  
 3 5  
 / \ / \  
7 8 1 3

Say two node be 7 and 1, result is TRUE.  
Say two nodes are 3 and 5, result is FALSE.  
Say two nodes are 7 and 5, result is FALSE.