جامعة الأميرة سمية للتكنولوجيا
PSUT
Princess Sumaya University for Technology

11212-Data Structures and Introduction to Algorithms

Assignment 3

Binary Search Tree (BST) Programming Assignment

Due Date: **Tuesday August 17, 2021, by Midnight**

Objectives

In this assignment, you will:

- practice the main concepts of a Binary Search Tree (BST)
- implement new operation on the BSTs according to given requirements

Submitting your work

- Deadline: Tuesday August 17, 2021 @ 11:59pm.
- All your work must be submitted through the eLearning portal
- Submit only the code your write (the source code of the functions you are asked to write)
- Make sure there are no compilation errors in your code.

Grading

[20 points] Modifed **`computeNodeDepths`**
[20 points] **`CountDeepNodes`**
[25 points] **`CountUnderVal`**
[25 points] **`allNodesFull`**
[10 points] Main Function

ABET-specific Note

This assignment addresses CLO2: Implement and use a wide range of data structures including basic sorting and searching algorithms.

Use the Binary Search Tree (BST) implementation given to you in this assignment to complete the following tasks.

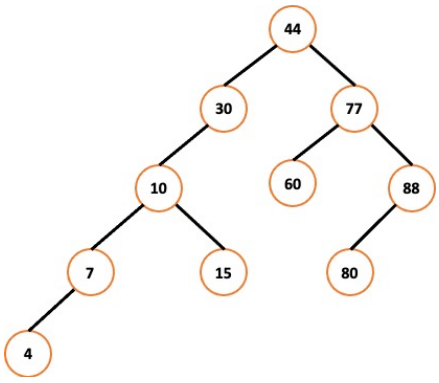
1. **[20 points]** In the **`computeNodeDepths`** implementation, instead of passing a pointer to the parent, we can simply pass the parent's depth (an integer). Rewrite the **`computeNodeDepths`** function such that it works correctly by passing the parent's depth. The function's header will become:

```
void BST<T>:: computeNodeDepths(TNode<T>* n, int parentDepth)
```

2. **[20 points]** Write the implementation of the **`CountDeepNodes(int x)`** member function, which takes an integer **`x`** and returns the number of nodes in the tree whose depths are larger than or equal to **`x`**. Assume that before running **`CountDeepNodes,`** all node depths have **already** been computed using the **`ComputeDepth`** function.

3. **[30 points]** Write the implementation of the member function **`int CountUnderVal(int val)`**, which searches the tree for a node whose value equals the parameter **`val`**. If such a node exists in the tree, the function returns the number of all nodes under the node **`val`**. If **`val`** does not exist in the tree the function returns -1. You may assume the tree values are unique.

   For example, in the tree below, calling the function with value 30 should return 4, calling the function with value 77 should return 3, calling the function with value 15 should return 0, and calling the function with value 100 should return -1.



4. **[30 points]** Write a BST member function **`allNodesFull()`** that will return true if each node in the tree has exactly 2 nodes, except for the leaf nodes, and returns false otherwise.

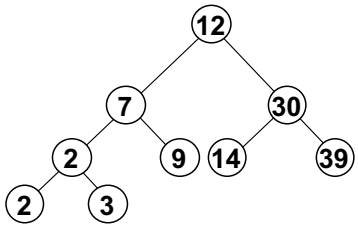


Figure 1: In this example, the function should return true as all nodes have exactly two nodes except for leaf nodes.

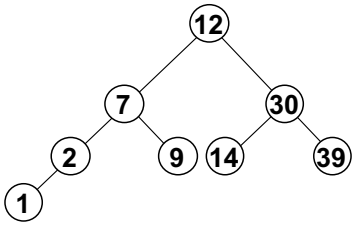


Figure 2: In this example, the function should return false since node 2 is not a leaf node but has only one node coming out of it.

5. **[10 points]** Write a main function that
   a. Declares at least one BST object
   b. Insert at least 10 values into your tree object (any integer or float values)
   c. Print the tree using one of the depth first traversal (DFT) methods (already implemented)
   d. Print the tree using a breadth first traverse (BFT) method (already implemented)
   e. **Test each of the functions you wrote** by calling them properly