2020-21
School of Computing, Engineering and Mathematics

5003CEM
Advanced Algorithms

Week 5
ASSESSED LAB TASK: ADV_1

Introduction (Recap)

5003CEM is assessed by coursework (10/15 credits) and exam (5/15 credits).

For the coursework, you need to submit a portfolio of work you've completed each week in the labs (and in your own time outside the labs).

Each week, there will be up to 4 types of task:

Non-assessed Standard	Assessed Standard
Non-assessed Advanced	Assessed Advanced

In Week 12, you'll need to submit ALL the assessed standard and assessed advanced tasks. There will be FIVE standard assessed tasks and THREE advanced assessed tasks. Assessed tasks will be evenly spaced across the module from Weeks 4 to 9.

There is no weekly deadline for assessed tasks; they are all part of your coursework, to be submitted on Friday 9 April,18:00. But it's a good idea to try to complete the assessed tasks in a timely way, so that they don't build up.

ASSESSED ADVANCED TASK 1/3

Assessed Advanced Task 1/3: Remove method for Binary Tree class **Advanced**

From the partial pseudocode given below (one case is omitted), implement an iterative method called remove which deletes a node and reorganises the tree. There are indications where the pseudocode is missing. NB the pseudocode crosses pages.

Add comments to show your understanding.

Implement your solution into the python Binary Tree class given on this week's Moodle in the zip folder, 'BST-class'.

Make sure that remove works correctly; that is, not only is the target node deleted, but the tree is also correctly re-organised.

You may decide, if you wish, to implement the entire class and solution in C++ instead of, or in addition to, the python solution, but this is not mandatory for this task.

```
REMOVE(tree, target)
    IF tree.root IS None                                    //if no tree
        RETURN False
    ELSE IF tree.root.data = target                         //if tree root is target
        IF tree.root.left IS None AND tree.root.right IS None
            tree.root ← None
        ELSE IF tree.root.left AND tree.root.right IS None
            tree.root ← tree.root.left
        ELSE IF tree.root.left IS None AND tree.root.right
            tree.root ← tree.root.right
        ELSE IF tree.root.left AND tree.root.right
            IF_LEFT_AND_RIGHT(tree.root)
```

(continues over)

```
                                                    //if root is not target
parent ← None
node ← tree.root

WHILE node and node.data != target
    parent ← node
    IF target < node.data
        node ← node.left
    ELSE IF target > node.data
        node ← node.right

IF node IS None OR node.data != target              //CASE 1: Target not found
    RETURN False                                    //for info only (we could not find it)

ELSE IF node.left IS None AND node.right IS None    //CASE 2: Target has no children
    IF target < parent.data
        parent.left ← None
    ELSE
        parent.right ← None
    RETURN True                                     //info only

ELSE IF node.left AND node.right IS None            //CASE 3: Target has left child only
    IF target < parent.data
        parent.left ← node.left
    ELSE
        parent.right ← node.left
    RETURN True                                     //info only

NOT IMPLEMENTED                                     //CASE 4: Target has right child only

ELSE                                                //CASE 5: Target has left and right children
    IF_LEFT_AND_RIGHT(node)
```

(continues over)

```
IF_LEFT_AND_RIGHT(node)                    //called if delete node whether root or otherwise
    delNodeParent ← node                   //has left and right children
    delNode = node.right

    WHILE delNode.left
        delNodeParent ← delNode
        delNode ← delNode.left

    node.data ← delNode.data

    IF delNode.right
        IF delNodeParent.data > delNode.data
            delNodeParent.left ← delNode.right
        ELSE
            delNodeParent.right ← delNode.right

    ELSE
        IF delNode.data < delNodeParent.data
            delNodeParent.left ← None
        ELSE
            delNodeParent.right ← None
```