UNIVERSITY OF THE WESTERN CAPE
COMPUTER SCIENCE DEPARTMENT
CSC211 DATA STRUCTURES – Term 2, 2020

TERM PROJECT

[Assessment weight: 40%]

Part 1: Binary Trees

1. Create a package called DataStructures
2. Create a class called Node with the following:
    a. A variable called value of type integer
    b. A variable called leftChild of type Node
    c. A variable called rightChild of type Node
    d. The constructor should accept an integer value and assign it to variable value. It should also set the leftChild and rightChild to null.
3. Create a class called BinaryTree
    a. Import the Node class created in 2 above
    b. Create a variable called root of type Node
    c. Create a constructor that accept an integer value and assign it as the root's value.
    d. Create a main method in which you should create a new instance of a Binary Tree and populate it with 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. Where 10 is the root, 20 is the root's left child, 30 is the root's right child, 40 is 20's left child, 5 is 20's right child, etc.
4. Write three methods to **recursively** traverse and print the BinaryTree using PreOrder, InOrder and PostOrder. As a guide (using PreOrder as an example):
    a. PreOrder (NLR): create a method called PreOrderTraversal which accepts a single parameter of type Node.
    b. If the argument received is null, then simply return
    c. Print out the value of the argument received (N)
    d. Call PreOrderTraversal and pass node.leftChild to it (L)
    e. Call PreOrderTraversal again and pass node.rightChild to it (R)
    f. Finally call PreOrderTraversal from the main method and pass root it.

    Repeat for inorder and postorder traversals.
5. Within you code, have a method or sequence that prints the output of the pre-, in- and post-order traversals.

Part 2: Binary Search Tree

1. Create a class called BinarySearchTree under the same package DataStructures
    a. Import the Node class created in part 1.
    b. Create a variable called root of type Node
    c. Create a constructor that assigns null to root.
2. Create a method called **add** that accepts an integer parameter.
    a. Create a new Node from the integer parameter (Node n = new Node(parameter))
    b. Compare the node with root,
        i. If node's value < root's: insert node into the left subtree
        ii. Else insert node into the right subtree.
    c. Hint: Recursion would be very helpful in achieving this
3. Create a method called **delete** that accepts an integer parameter.

    a. Create a new Node from the integer parameter (Node n = new Node(parameter))
    b. Repeatedly compare n with nodes on the tree
        i. If node's value < root's: delete node into the left subtree
        ii. Else delete node into the right subtree.
    c. Recall the rules of deleting from BSTs, if the node to be delete…
        i. has no child, then replace it with null
        ii. has 1 child, replace it with the child
        iii. has 2 children, replace it with the smallest child in its right subtree
4. Create a method to find an element in a Binary Search Tree
    a. Name the method **findElement** and it should accept an integer parameter.
    b. Recursively search through the BST for the integer
5. Copy your preOrderTraversal, inOrderTraversal and postOrderTraversal methods from part 1 into this class.
6. Create a Main method
    a. Make an instance of the BinarySearchTree class
    b. Manually populate the BST with the following 14, 3, 4, 12, 13, 15 and 10. Do this by using BST.root = new Node(14); BST.root.leftChild = new Node(3)…
    c. Call the pre..., in... and postOrderTraversal methods to print out the BST as it is.
    d. Using the add method you created, add the following into the BST 5, 1, 8, 2 and 0.
    e. Call the three traversal methods to print out the BST after the insertions
    f. Using the delete method you created, delete 0 and 15 from the BST.
    g. Call the three traversal methods to print out the BST after deleting
    h. Accept an integer variable from the user.
    i. Using the findElement method you created, check if the variable entered by the user is present on the BST. If present return FOUND, else return NOT FOUND.

Part 3: Binary Heap

1. Create a class called BinaryHeap under the same package DataStructures and import the Node and BinaryTree classes from Part 1.
2. Within the BinaryHeap class:
    a. Create an integer array named heap
    b. Create a variable of type BinaryTree (e.g. BinaryTree bt)
3. Create a constructor for the BinaryHeap class that accepts at integer variable called elementCount as a parameter. Within the constructor, instantiate the heap array and set its size to elementCount
4. Create a method called TreeMaker

a. Instantiate the BinaryTree created in 2b above (bt = new BinaryTree())

b. Create a BinaryTree with 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. This is similar to what you did in Part 1, question 3d.

5. Implicitly represent the BinaryTree created in 4 above in the heap array (created in 2a above). See slides 15, 23 and 24 of the Powerpoint lecture slide on PriorityQueues
6. Create a method called getParentPosition(), which accepts an integer parameter (index of a node on the heap) and returns an integer representing the position of its parent node).
7. Create two methods called getLeftChildPosition() and getRightChildPosition(). These two methods should accept an integer (representing the index of a node on the heap) and also return an integer (representing the position of the left or right child respectively).
8. Create a method called insertNode(), this method should:
    a. accept an integer

    b. insert the value into its appropriate position the heap – bubble up. See slides 16 – 26 of first lecture on Priority Queue.
    c. Print out the tree (pre-, in- and post-order traversal) before and after insertions
9. Create a method called deleteNode(), this method should:
    a. accept an integer value
    b. Check if the value exists on the heap
    c. If it does, delete it from the heap
    d. Replace it with the appropriate replacement node. See lecture on deleting from BinaryHeap.
    e. As a hint, you need to create a method to find the minimum between possible alternatives. The ternary operator (?) in Java could come in handy here.
    f. Print out the tree (pre-, in- and post-order traversal) before and after insertions.
10. You should of course create a main method within this class..

Note:

1. Since these classes have similar methods, you can use OOP and simply make both your Binary Search Tree and BinaryHeap classes extend the Binary Tree class. You can then reuse your existing methods and override where necessary. **This is optional and you would not be penalized if you do not do it this way.**

2. For each of the three classes, include instructions in your code to print out your information when each program starts. The header should contain the following:

> Name: SURNAME, Other names
> Student Number: XYZ123
> Course Code: CSC211_Term Project 2020

This is compulsory! After the header, then your code can begin normal execution.

3. As with all assignments / projects, it is in your best interest not to plagiarize. The Computer Science department and UWC in general does not take cases of plagiarism likely.

Submission

1. You would be expected to submit your full source codes. These can be Java files or codes pasted into MS Word.
2. Snapshots of the output of your code, this should simply be a screen capture (print screen) of your screen showing the output of your executed codes.