COSC 3333 - Data Structures & Algorithms II **Spring 2023**

Programming Assignment 3 **Due Date: Saturday, April 15 @ 11:59 p.m.**

Download the code provided by the textbook in order to complete this program.
Please note that no Java classes for specific data structures or associated algorithms can be used for any assignments for this course, except the one in `Tree` class that is used in `displayTree` method.

You MUST follow the outlined instructions to secure the full credit. No credit will be given to programs that just generate the output but do not follow the methodology given.

The assignment is based on a complete or nearly complete Binary Tree, where each node carries a numeric label, as well as a random uppercase alphabetical letter. The user gets to choose the size of this Binary Tree in the range 5-20 inclusive.
You will build it by numbering each node sequentially, and populating its character attribute by a random uppercase alphabetical letter.

The following gives an example of a 15-node and a 9-node BT:

```
                                1
                    2                       3
            4               5               6               7
        8       9       10      11      12      13      14      15
```

```
                                1
                    2                       3
            4               5               6               7
        8       9       --      --      --      --      --      --
```

As you can see, these nodes are sequentially numbered and are positioned based on their numeric labels.
So for the first step of the assignment build this Binary Tree. Notice this BT will NOT be built based on the letter attributes of the nodes. This must be a complete or nearly complete Binary Tree whose nodes' numeric labels are based on the position of the node on each level of the Binary Tree. How do you think you can build such tree?

Whatever data structure or methodology you used to build your Binary Tree, after completion of it, you are only going to refer to this Binary Tree for the rest of the operations for this assignment and not the structure(s) you used to build it.
It is of utmost importance that your code adheres to these specifications.

One task of this assignment involves the user choosing nodes only based on their numeric labels and see if that combination results in any meaningful word which will be composed of each

node's letter attribute. Notice we will not be able to use any AI for that determination, it will be for the user to judge the resultant word! The user can choose nodes as many times as they wish. However, they must first determine how many letters/nodes they will be choosing. Next they refer to the numeric labels of the nodes, where you need to look each node up based on their positioning numeric label, and display their letter attribute one next to another to form a word. You may also choose to append these letters to form as string.

As you already know, Binary Trees are not Search Trees, so even if we know the numeric label of a node, we still need to check every node or do some arithmetic to get to that node, every time the user selects a node.

In the textbook, we covered a strategy in finding the last node in a Tree-based Heap. Here we are going to use that technique along with the hashing scheme of *direct addressing* to facilitate the search for any given node based on their position in the tree i.e. their numeric label.

The hashing scheme hashes each node based on their label to the matching hash index. Node #3 hashes to index 3 in the hash table. Notice that the reasonable size of our Binary Tree makes it possible for us to use direct addressing. But since these nodes are id'ed by their numeric labels and not any other attributes, we need to store the path to get to them, in their hash index and not say their letter attribute. What size do you choose for the hash table?

So in summary: After successfully building this Binary Tree, you need to go through each node and use the textbook's technique to come up with their path to store in the proper hash index in the hash table. This is so that for example when the user picks node #13 we pick up the "routing" sequence of "101" which takes us: From the root, go one right, one left, and one right to get to node #13 for whatever purpose.
The textbook gives us the loop version of this technique, you MUST write it RECURSIVELY.

After hashing all the nodes's paths based on their numeric labels. You are ready for the following menu:

```
1.    Display the Binary Tree(numeric labels)
2.    Display the Binary Tree(letters)
3.    Build a Word:
      Enter the size of the word:
      Input the nodes' numeric label(s):
4.    Add New Node
5.    Display the nodes on a level/depth:
      Enter the level/depth
6.    Clear letter:
      Enter the node's numeric label:
7.    List the vacant nodes
8.    Display the word formed at the last level
9.    Exit
```

Option 2. In case the letter attribute of the node has been "cleared" as a result of option 6, display `[ ]` to represent no letter.
Option 4. You must validate user input: the total number of nodes should not exceed 20. This option should be followed by hashing the new nodes. New nodes can only be added at the end of the tree, the first available spot in a nearly complete or complete Binary Tree.

Option 5. You must validate user input for a valid level/depth. Most importantly you MUST use the hash table for this option. You have already done the tree version of this in the previous assignment. Which one is easier?

Option 6. Finds the node and simply assigns space for the letter to be deleted. Within our assignment this will have the connotation that the node is vacant.

Option 7. Must use the hash table for this option as well. Must only list the nodes that were "cleared". You MUST not only refer to them by their numeric values but need to mention how many and on which level, for example:

```
The vacancies:
#3 : 1 on level 1
#10  #14 : 2 on level 2
```

Option 8. Must use the hash table for this option as well. You can simply display the letters of the nodes residing at the last level of your Binary Tree.

I would explain the assignment further in class: Do not miss the next session. Don't miss any sessions!

Happy Programming!

- *The program must be written in Java.*
- *You must upload all relevant .java files, including the ones from the book, on blackboard to ensure successful execution of your code, by the due date.*
- *The submission that is missing required files/classes will be considered void.*
- *Programs that do not compile will be considered void.*
- *This is an individual assignment. If students submit fully or partially identical programs, all individuals involved will earn a grade of zero.*