COSC 3333 - Data Structures & Algorithms II **Spring 2022**
Programming Assignment 6 **Due Date: Thursday, May 12 @ 11:59 p.m**

Topic: Graphs
Download the code provided by the textbook in order to complete this program.

We all know by now that any acyclic connected graph can be considered a tree. Write a program that for the purpose of practicing with these structures "maps" a BST to a Graph.

In previous assignments you have read a word (a `String`) from the keyboard, dissected it into its single letters and configured these letters into a Binary Search Tree. (The word will be all in Caps.)You are going to start with the same process here. But then you will take your BST and store it in a Graph.

This means that you should visit every node in the BST and not only store that as a vertex in the array that represents the Graph, but also populate the adjacency matrix based on the parent-child relationship that you detect in your BST.

An important point to remember is that the order of the vertices in the vertex array does not tell us anything about the connectivity of these vertices. It's all about the adjacency matrix that determines the connectivity pattern of the graph.

Add the following methods to the code given in the textbook:

```
displayVertexList()
```

```
displayAdjMatrix()
```

Make sure this method displays the matrix in a tabular format with rows and columns in an organized manner for readability.

We are going to let the user decide whether they want this to be a *directed* or *undirected* graph. So the next question to the user would be the following:

```
Map the BST into:
1.    Directed Graph
2.    Undirected Graph
```

Please note that in case of the user choice for directed graph, we use the parent-child direction: from parent to the child.

After mapping the BST into the proper Graph, you will display the following menu and let the user choose from the options:

```
1.    Display the BST in a tree format.

2.    Display the Vertex array.
```

```
3.    Display the Adjacency Matrix

4.    BONUS: Given a vertex: Display ALL possible separate paths starting
      with that vertex in a Depth First Search pattern:
      Enter the letter:

5.    Given a vertex: Display ALL its adjacent vertices (one edge apart)
      Enter the letter:

6.    Given a vertex: Display ALL the vertices that are two edges away from
      it:
      Enter the letter:

7.    Exit
```

A Note: You start this program by reading the user input string into a BST, you should hold on to your BST object, in case at any point the user chooses option 1 which is to display this tree as a visual tree.

But any other operations other than one on option 1, MUST BE DONE ON GRAPH VERSION OF THIS TREE. For example in order to find the adjacent vertices of a given vertex YOU CANNOT TRAVERSE THE BST, YOU SHOULD USE THE ADJACENCY MATRIX.

PLEASE BE ADVISED THAT ANY TREE OPERATION IS GOING TO BE CONSIDERED VOID FOR THIS PROBLEM. THIS IS NOT A TREE PROGRAM!

Here are additional explanations of some of the menu options:

Option 4. Here is an example of this option referring to figure 13.5 on page 625 in the textbook (acknowledging that our graph will be based on a BST and will look differently than this figure): If the user enters letter `A`, The following would be ALL the paths in a depth first search pattern that starts with `A`:

```
ABFH
AC
ADGI
AE
```

Option 5. Referring to the Figure 13.5 example again: if the input is letter `G`, the one edge apart vertices would be `D` and `I`.

Please note that for options 4 and 5, you should modify the code in the book so that you can specify the index of the vertex you would like to choose as starting point. Currently the code starts at index 0 (hard coded for this value.) you need to change that. Again you can safely assume the user input will be valid.

Option 6. Again referring to the above example: given vertex `A`, the vertices two edges away would be: `F` and `G`. Obviously in case no vertices fit the criteria, you should display a message accordingly.

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.*