Practical-06 Part II: Tree and Graph

Due 24 Oct by 22:00 **Points** 100 **Available** after 4 Oct at 0:00

You should use JAVA to complete this practical.

Submission

Note: all class work must be submitted to your SVN repository. For this practical, you shall use:

```
https://version-control.adelaide.edu.au/svn/<YOUR-ID>/2019/s2/fcs/week-10/practical-06
```

Part 1 - Binary Search Tree

Question 1: Set up a new project in your IDE with a Main class. Your project name should be BinarySearchTree. All your files for this part should be saved under **practical-06/BinarySearchTree/**. You can look at this if you are not familiar with binary search trees.
video (https://youtu.be/4o9vzzTxbs8)

(https://youtu.be/4o9vzzTxbs8)

Question 2: A **Binary Search Tree** is a node-based binary tree data structure. Define a **Node** class to represent the elements in the tree. The **Node** class should have a **data** field saving the data contained in this node, a **left** field and a **right** field that refer to its left child node or right child node, respectively. In the constructor, **left** and **right** should be initialised as **NULL**.

```
public class Node
Attributes:
private int data; // the data saved in this node
private Node left; // the reference to its left child
private Node right; // the reference to its right child

Methods:
// Print the data saved in the node.
public void printNode();

// You should also implement the constructors, mutators and accessors.
...

Example output for printNode():
Node data: "<data saved in this node>"
```

Question 3: Define a **MyBST** class. The **MyBST** class should have a **root** field which saves the **root node** of this tree. You need to

- Implement an **insert(int value)** method to insert a value into this tree. To insert a value into a tree, you need to start by comparing the inserting element data with the root node data, if the value is less than the root value, then recurse for the left child, else recurse for the right child. After reaching the **leaf node**, insert the value at the left child (if the value is less than the leaf node value) else, insert at the right child. *Note: If a node is already in this tree, do not insert it again, instead, print out "Node is in the tree".*
- Implement a **search(int value)** method to search for a value in this tree. To find a value in a tree, you need to start with comparing the search value with root's nodes value, if the value is equal to the root, then return **TRUE**, if it is less than the root, then recurse for the left child, else, recurse for the right child. After reaching the leaf node, if the value is not equal to the leaf node's value, return **FALSE**.

```
public class MyBST
Attributes:
private Node root; // The reference to the root node in this tree

Methods:
// You should initialise an empty tree in the constructor.
public MyBST();

// Insert a new value into the tree. This method calls insertRec()
public void insert(int value);

// This is a recursive function to insert a new value in the tree.
private void insertRec(Node current, int value)

// Search a node in the tree. This method calls searchRec()
public boolean search(int value);

// This is a recursive function to search a node in the tree.
private boolean searchRec(Node current, int value)
```

Part 2 - Graph

Question 4: Set up a new project in your IDE with a Main class. Your project name should be **Graph**. All your files for this part should be saved under practical-06/Graph/

Question 5: Graph consists of **nodes** that can be connected to each other. Define a **Node** class to represent the nodes in a graph. The **Node** class should have an **index** field saving the index of this node. Each Node will have a different index in Question 6.

```
public class Node
Attributes:
private int index; // the index of this node

Methods:
// Print the data (the index) saved in the node.
public void printNode();

// You should also implement the constructors, mutators and accessors.
...

Example output for printNode(), you should replace the <index> with the value saved in index:
"Node <index>"
```

Question 6: Define a **MyGraph** class. An **Adjacency Matrix** and **Adjacency List** are the most commonly used representations of a **directed graph**.

In this question we will refer to each **Node** by it's **index**. These indexes will be referred to by ***i*** and ***j***.

An **adjacency matrix** is a **2D array** of size V*V where V is the number of vertices (nodes) in the graph. Let the 2D array be **adj[][]**, a element adj[i][j] =1 indicates that there is an edge from node i to node j, and adj[i][j] = 0 indicates that there is no edge from node i to j.

An **adjacency list** is an **array of linked lists**. The size of the array is equal to the number of nodes. Let the array be **array[]**. An entry array[i] represents the list of nodes adjacent (connected) to the *i* th node. If node j is in the list of nodes saved in array[i], there is an edge from node i to node j, otherwise, there is no edge from node i to node j.

In this question, you are asked to implement a **matrixToList()** method to transform an adjacency matrix into an adjacency list. **Note:** You can use the **LinkedList (https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html)** class in Java instead of building a Linked list by yourself.

```
public class MyGraph
Attributes:
LinkedList<Node> adjListArray[];

Methods:
// You should initialise an empty graph in the constructor.
public MyGraph();

// transform an adjacency matrix to an adjacency list
public void matrixToList(int[][] matrix);

// Print out the adjacency list array
public void displayAdjListArray();

Example
Input matrix (The first row and the first column are the index of the 2d array to help you understand. They are not a part of input):
  0 1 2 3
0 0 1 0 1
1 1 0 0 0
2 0 0 0 1
3 0 1 1 0

Output for displayAdjListArray:
0: Node 1 -> Node 3
1: Node 0
2: Node 3
3: Node 1 -> Node 2

Hint: You should call printNode() method defined in question 5 to help you display the adjacent list array.
```