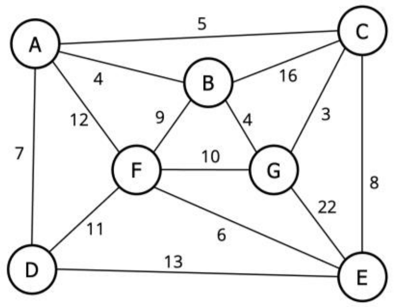
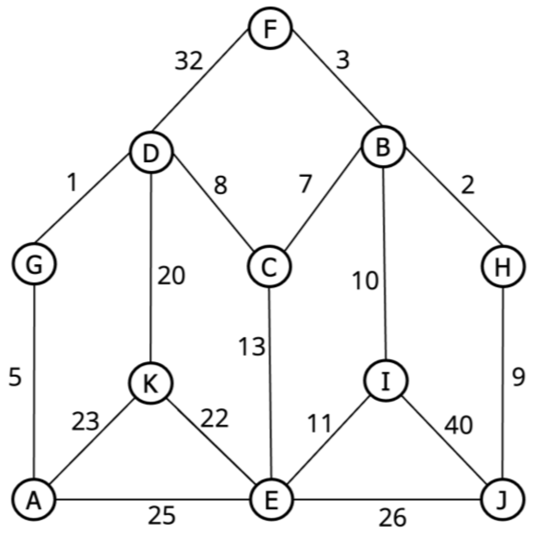
COSC 222: Data Structures
Lab 7 – Graphs

Question 1 [8 marks]: Question 1 does not require coding. You will need to submit a PDF file with your answers. This can be handwritten and scanned, drawn on a tablet, or created using a diagramming program and word processor. Name your file **Lab7Question1.pdf**.

A. Prim's Algorithm [4 marks]: Apply Prim's algorithm on the graph to construct a minimum spanning tree starting with vertex A. If there are any ties, the vertex with the lower letter comes first (for an example, unvisited vertex 'B' should come before another unvisited vertex 'C'). Write the **vertices** and **edges** in the order in which they are added to the tree. Also, include the **total cost** to span the tree. Show each step (feel free to use graphs.docx file for showing each step).



B. Kruskal's algorithm [4 marks]: Step through Kruskal's algorithm (covered in lecture 14) to calculate a minimum spanning tree of the graph. If you can select two edges with the same weight, select the edge that would come alphabetically first (e.g., select (B, C) before (E, F) or (A, B) before (A, F). Also, when representing an edge, arrange two letters in alphabetical order, e.g., use (C, E) instead of (E, C). List the **edges** in the order in which they are added to the tree. Also, include the **total cost** to span the tree. Show each step (feel free to use graphs.docx file for showing each step).



Question 2 (Graph Representation): Adjacency Matrices and Lists [10 marks]
While we typically visualize graphs as interconnected webs of nodes, graphs in code are typically represented in 2 ways: adjacency matrices and adjacency lists. These three formats are shown below in Figures 1-3.

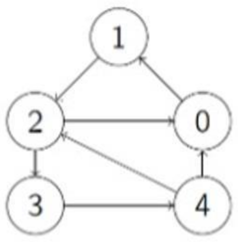


Figure 1: Graph

	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	1	0	0	1	0
3	0	0	0	0	1
4	1	0	1	0	0

Figure 2: Adjacency Matrix

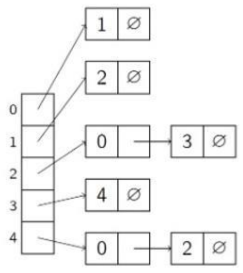


Figure 3: Adjacency List

In this question, you will write a program which will create a graph and then print it as both an adjacency matrix and as an adjacency list. You will need the following files: **Graph.java** and **GraphTest.java**.

First review **GraphTest.java**. You do not need to change anything in this file. Notice the method `randomMatrix()` in particular. This method generates a random double-array of integers, with the size given, representing the adjacency matrix of a graph. This random double array is then taken as the only argument for the **Graph** constructor. Your task is to complete all of the necessary methods within the **Graph.java** file.

Part A [5 marks]:
`Graph()` and `generateAdjList()`: First write the constructor of the graph. This constructor takes in the adjacency matrix (int[][]) as its only argument. It then initializes the Graph (i.e., initialize the values of numVertices and adjMatrix). You should also call `generateAdjList()` from the constructor. This method takes no arguments, but uses the data in the newly saved `adjMatrix` to populate the `adjListArray`.

Note: Please use Java's default LinkedList class. Here is a sample code showing how to add four number into an array of LinkedList. If you prefer to use a list instead of an array, feel free to make corresponding changes in the code.

```
import java.util.LinkedList;
... ... ...
int n = 2;   //Array size 2
LinkedList<Integer>[] arrayLinkedList = new LinkedList[n]; //Initialize array
arrayLinkedList[0] = new LinkedList<Integer>();     //Initialize array elements
arrayLinkedList[1] = new LinkedList<Integer>();     //(objects of LinkedList)

arrayLinkedList[0].add(101);     //Add 101 and 102 to arrayLinkedList[0]
arrayLinkedList[0].add(102);
arrayLinkedList[1].add(201);     //Add 201 and 202 to arrayLinkedList[1]
arrayLinkedList[1].add(202);
```

```
for (int i = 0; i < n; i++) {    //Showing the elements with get() method, you can also
    for (int j = 0; j < arrayLinkedList[i].size(); j++) {  //use for-each loop
        System.out.print(arrayLinkedList[i].get(j) + " ");
    }
    System.out.println();
}
Output:
101 102
201 202
```

Part B [2 marks]:
`printMatrix()`: This method takes no arguments, but prints the adjacency matrix (`adjMatrix`) in the format shown below. Note that since the graphs are randomly generated, the values will not be identical to the output shown.

Part C [3 marks]:
`printList()`: This method takes no arguments, and prints the adjacency list (`adjListArray`) in the format shown below. Note that since the graphs are randomly generated, the values will not be identical to the output shown.

Practicing drawing the graphs generated by `randomMatrix()` may be a good exercise for your exam. Try drawing the graphs first by looking at either the matrix or the adjacency list and then checking from the other.

Sample Output:

```
Graph 1:
Adjacency matrix (4 nodes):
   0 1 1 1
   0 0 0 1
   0 0 0 1
   1 0 1 0
Adjacency list of vertex 0: 1, 2, 3
Adjacency list of vertex 1: 3
Adjacency list of vertex 2: 3
Adjacency list of vertex 3: 0, 2

Graph 2:
Adjacency matrix (7 nodes):
   0 0 1 0 0 0 1
   0 0 1 1 1 1 1
   0 1 0 0 0 1 0
   1 0 1 0 0 1 1
   0 1 1 1 0 1 1
   0 0 1 1 1 0 0
   0 0 0 0 1 0 0
Adjacency list of vertex 0: 2, 6
Adjacency list of vertex 1: 2, 3, 4, 5, 6
Adjacency list of vertex 2: 1, 5
Adjacency list of vertex 3: 0, 2, 5, 6
Adjacency list of vertex 4: 1, 2, 3, 5, 6
Adjacency list of vertex 5: 2, 3, 4
Adjacency list of vertex 6: 4
```

Question 3 (DFS): Count Starting Nodes [2 + 3 (bonus) marks]

This question asks that you write a program which will take an undirected graph and visit all the vertices using a Depth-First Search (DFS). However, with some graphs it is not possible to reach every vertex from one starting vertex (i.e. a disconnected graph). Therefore, you may need to select more than one starting vertex to complete the graph traversal.

Begin with the files **DFSGraph.java** and **DFSTest.java**. In **DFSTest.java**, one of several unique graphs is generated. Then two methods from the **DFSGraph.java** class are called, which you will need to complete.

Part A [2 mark]:

`printList()`: This method prints out the graph in an adjacency-list format. You may adapt your `printList()` method from Question 2.

Part B [3 marks (BONUS)]:

`countStartingNodes()`: This method finds the number of vertices that need to be selected as starting vertices to traverse the entire graph. Check Lecture 12 slide 24. For the left graph, we can select one vertex and traverse the entire graph. However, for the right graph, you need to select minimum two vertices to traverse the entire graph. Note that you must select vertices in ascending order (i.e. you must try starting from vertex 1 before trying vertex 2). Hint: you may want to use a boolean flag to keep track of which vertices have already been visited.

Within this method, you should call `DFS()`. This method traverses the graph from the given starting vertex, maintaining a record of which nodes have been visited. Recursion should be used with this method. *Note that you may wish to change the return type.*

When the traversal has been completed, `countStartingNodes()` should print the number of starting vertices selected and a list of the selected vertices.

If you complete these methods correctly, you should see the following output when testing your program with the provided `graph0` matrix.

Sample Output:

```
Adjacency list of vertex 0: 1
Adjacency list of vertex 1: 0, 2
Adjacency list of vertex 2: 1
Adjacency list of vertex 3: No vertex found

You can traverse the entire graph by selecting 2 vertices
Starting vertices are: 0 3
```

Submission Instructions:

- Create a folder called "Lab7_<*student_ID* >" where <*student_ID* > is your student number/ID (e.g. **Lab7_12345678**). Only include the mentioned java files in the folder. **DO NOT** include any other files (e.g., .class, .java~ or any other files)
    - For Question 1, include a pdf file your **Lab7Question1.pdf** file.
    - For Question 2, include your **Graph.java** file.
    - For Question 3, include your **DFSGraph.java** file.
- Make a **zip file** of the folder and upload it to Canvas.
- To be eligible to earn full marks, your Java programs **must compile and run** upon download, without requiring any modifications.
- These assignments are your chance to learn the material for the exams. **Code your assignments independently.**