Programming Project 6 – Graph Application and Implementation

Note: When you turn in an assignment to be graded in this class, you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from your instructor). Any assistance from other sources, including the internet, is an honor code violation.

In this project you will extend the UnweightedGraph class by implementing a class called MyGraph that adds functionality for the following five methods:

Test whether a graph is connected

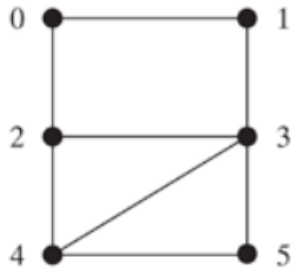
1. Add a method called **`isGraphConnected`** that reads a graph from a file and determines whether the graph is connected.
    - The header for this method is

      ```
      public boolean isGraphConnected(String fileName)
      ```

    - Your method will accept a file name as a String parameter and if the file is unable to be opened, the method should throw a FileNotFoundException
    - The first line in the file contains a number that indicates the number of vertices (**n**).
    - The vertices are labeled as **0**, **1**, …, **n–1**.
    - Each subsequent line, with the format **u v1 v2** …, describes edges (**u**, **v1**), (**u**, **v2**), and so on.

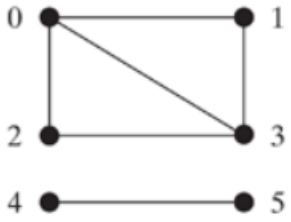
   Here are examples of two files for their corresponding graphs:

```
File
6
0 1 2
1 0 3
2 0 3 4
3 1 2 4 5
4 2 3 5
5 3 4
```



(a)

```
File
6
0 1 2 3
1 0 3
2 0 3
3 0 1 2
4 5
5 4
```



(b)

Hint: The method should read data from the file, create an instance **g** of **UnweightedGraph**, invoke **g.printEdges()** to display all edges, and invoke **dfs()** to obtain an instance **tree** of **UnweightedGraph.SearchTree**. If **tree.getNumberOfVerticesFound()** is the same as the number of vertices in the graph, the graph is connected.

Find connected components

2. Add a method called **`getConnectedComponents()`** for finding all connected components in this graph instance with the following header:

   ```
   public List<List<Integer>> getConnectedComponents()
   ```

    - The method returns `a List<List<Integer>>`. Each element in the list is another list that contains all the vertices in a connected component.

Find paths

3. Add a method called **`getPath`** for finding a path between two vertices with the following header:

   ```
   public List<Integer> getPath(int u, int v)
   ```

- The method returns a `List<Integer>` that contains all the vertices in a path from **u** to **v** in order.
- Using the BFS approach, you can obtain the shortest path from u to v.
- If there isn't a path from u to v, the method returns null.

Detect cycles

4. Add a method called **`isCyclic()`** for determining whether there is a cycle in the graph with the following header:

```
public boolean isCyclic()
```

- The method returns true if there is a cycle in this instance of MyGraph.

Find a cycle

5. Add a method called **`findCycle`** that will find a cycle in this graph that starts at vertex **u** with the following header:

```
public List findCycle(int u)
```

- The method returns a `List` that contains all the vertices in a cycle starting from **u**
- If there is not a cycle in this instance of `MyGraph`, the method returns **null**.

Your file must be written in package – **`cmsc256`** .Upload both the **`MyGraph<V>`** class and the **`MyGraphTest`** class to Gradescope for grading.

Follow the Coding Style Guideline in Blackboard and include a comment block that includes your name, the name of the project along with the file name and a brief description of the purpose of the class at the top of **the source code file that you submit**.

Ask questions about any part of the programming project that is not clear.