CSCI 230 – Final Project

Collaboration policy: **Individual Assignment**
Total Points: **100**

Source Code

The Java classes provided in the zip file attached to the project Dropbox are:

- `Vertex.java`
- `UndirectedGraph.java`
- `VertexException.java`

Under no circumstances are you allowed to modify or create a new `Vertex` or `VertexException` class. You must use these files **as is**.

`ConstantTimeStack`, `ConstantTimeQueue`, `ArrayList`, `DoublyLinkedList` classes developed in previous homework assignments are reused in this project assignment. Under no circumstances are you allowed to modify these classes, you must use these files **as is**. Furthermore, you may use the solutions provided on OAKS.

The `UndirectedGraph` class will be modified by you. In particular, the methods listed in Part 1 are to be modified. Furthermore, you may add additional method and/or private instance variables to help support the two new methods listed in Part 1. **However**, you must use in your solution: 1) the adjacency list provided in the `UndirectedGraph` class, 2) the provided `Vertex` class, and 3) you may not create additional classes. If in doubt about these restrictions -> you must ask!

Lastly, you **may not** change the package structure! Specifically, `edu.cofc.csci230` cannot be removed or modified. If a solution is submitted with a different package structure, it will not be graded, no exceptions.

Part 1

In the `UndirectedGraph` class please fully implement the methods listed below:

- `public Boolean isConnected()`
- `public String findAllCycles()`

In each method listed above you will see a TODO comment, this is where your coding solution is added. In the provided source code, numerous comments are given; please ensure you read them carefully. Additionally, in the course textbook and supplemental textbook the two topics (along with definitions) cover these topics.

Part 2

The provided `UndirectedGraph` class has a main method. In the main please add test cases that demonstrate you have fully evaluated the operational correctness of the methods you implemented in Part 1. To receive full credit, these test cases **must** be included.

Plagiarism

This is an individual project and plagiarism WILL NOT be tolerated. If your work is not your own, i.e. copied from another student or the web, you will be given a zero the project and turned in for a honor code violation. I assure you, I you do your own work, everything will be OK.

Submission

Create a zip file that **only** includes the completed `UndirectedGraph.java` file. If you have any questions about the submission policy, you must resolve before the due date. Lastly, please plan appropriately, asking questions the day the project is due (within 12 hours) is too late. Please try to resolve any questions at least 2 days before the due date.

The name of the zip file must be your last name. For example, *ritchie.zip* would be correct if the original co-developer of UNIX (Dennis Ritchie) submitted the assignment. Only assignments submitted in the correct format will be accepted (no exceptions).

Please submit the zip file (via OAKS) to the Dropbox setup for this assignment by the due date. You may resubmit the zip file as many times as you like, Dropbox will only keep the newest submission. Per the syllabus, late assignments will not be accepted – no exceptions. Please do not email Hassam or I your assignment after the due date, we will not accept it.

Grading Rubric

UndirectedGraph Compiles	15 points
Thoroughness of your test cases	5 points
Instructor test cases (8 cases 10 points each)	80 points
	100 points

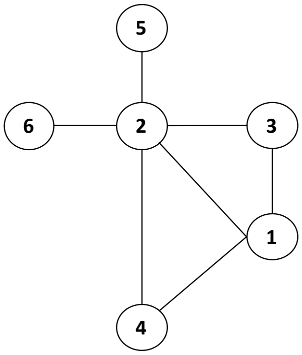
In particular, each data structure will be graded as follows. If the submitted solution

- Does not compile: 0 of 100 points
- Compiles but does not run: 15 of 100 points
- Thoroughness of your test cases: 20 of 100 points
- Passes all 8 test cases developed by instructor: 100 of 100 points.

Example Graph Operations and Search Output Format

To get you started, the main method comes with code to help understand how the undirected graph class is instantiated and used. Furthermore, example depth and breadth first search methods are provided that may be used to guide you in this project.

For those that are more visual, please see the examples below.



Example Java Code

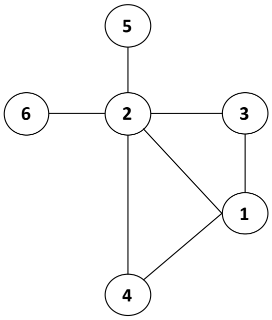
```
UndirectedGraph<Integer> graph = new
UndirectedGraph<Integer>();

graph.addEdge( 1, 2 );
graph.addEdge( 1, 3 );
graph.addEdge( 1, 4 );
graph.addEdge( 2, 3 );
graph.addEdge( 2, 5 );
graph.addEdge( 2, 6 );
graph.addEdge( 2, 4 );

System.out.println( graph );
```

Example Output from toString() method

```
Vertex (1): [ 2, 3, 4 ]
Vertex (2): [ 1, 3, 5, 6, 4 ]
Vertex (3): [ 1, 2 ]
Vertex (4): [ 1, 2 ]
Vertex (5): [ 2 ]
Vertex (6): [ 2 ]
```



Example Java Code

```
System.out.printf( "DFS\n%s\n", graph.depthFirstSearch( 1 ) );
System.out.printf( "BFS\n%s\n", graph.breadthFirstSearch( 1 ) );
```

Example output format from BFS and DFS method calls

```
DFS
1:1
2:3
3:6
4:2
5:5
6:4


BFS
1:1
2:2
3:3
4:4
5:5
6:6
```