CMSC 330 Project 1

Burkhardt, Ryan

UMGC CMSC 330

Author Note

This program takes the users input in the form of a mouse click, button or text to create a graph GUI and allow the user to interact with the graph. It includes the methods to find out if it is connected, has cycles, perform a depth first search and a breadth first search of the tree the user creates.

UML Class Diagram

A screenshot of a computer program

Description automatically generated

**Assignment 4**

For this Assignment we were given the task of creating a graph that the user could interact with in multiple ways. We were also assigned stipulations such as having 4 classes and specific methods in each class. The concept of the graph reminded me a lot of the Binary Tree methods we have been learning about in the past two weeks. However, I did enjoy how I could figure out most of my problems without recursion this time. With such a hard concept to grasp I did attempt a small amount of recursion and left it in a test folder. This week’s assignment was a great challenge and I enjoyed learning about the HashMap’s. This is truly a tool I wished we learned about much earlier in the Computer Science course.  
 The **Vertex Class** is a simple class that defines a Vertex object to include the double values for the x and y coordinates. It also gives the vertex a char as the name which we will use in the HashMap as the key. Lastly it includes a toString() method to state the name of the vertex that was created and its location.

The **Graph Class** was the next class and the most challenging in my opinion. In this class I defined two Hash Maps for the vertices and the edges. These would be used to store the vertices and edges the user creates. I then initialize a list of characters from A-Z to use as my names which I will index in the addVertex method. This method uses an index variable to place the name as a char from our list and then using the put method to add it to our hash map. The getVertex() takes in a character and runs it through a for loop to find a vertex with that name and return that vertex. The addEdge() is identical to the addVertex() but instead of returning a vertex it returns an edge made of the integer index as well as the two vertices connecting the edge. The getConnectedVertices() and getConnectedEdges() use very similar for loops to return a list of connected vertices and edges. The checkCycle uses recursion with the public method calling its private method. With this recursion it will check a specific edges vertex to see if either vertex has an attached edged that will eventually lead back to one of its original edges. It will then call itself recursively to check every edge in the list. The is connected I found an implementation with a nested for loop to visit each vertex on the edges and only add it to the visited list if it was unique. Then after it has checked all the edges vertices it compares the size of visited list to the size of vertices returning true if they are equal. depthFirstSearch() utilizes the stack method very similar to our text but using a Vertex object instead of the int[][] in the text. The example from Dr. Liang helped me model this appropriately. The same can be said for my breadth first search using the queue. I took both examples and tailored them to model the vertices and traverse the edges with a while loop. Both searches I added the implementation from the instructions to specifically start with ‘A’ by calling the .get() key method in the hash map. Lastly in my graph page I also established the Edge class and object using the getters to get the starting and ending vertexes of my edge as well as the name getter from the vertex class to implement the name as the toString() method within. I found this not required but necessary for traversing my graph.

The **P4Pane Class** I found to be the most challenging. This was mainly due to me trying to tailor the methods I built into the graph class and call them here but modify them every so slightly to draw the vertices and edges. I modeled most of my FX directly from Project 2’s point pane classes. I found these to be very helpful in this assignment. I added all buttons, text fields, and labels as necessary and all of their actions to add the vertex and edge to the graph. I also added a small amount of customization so the user could not add vertices where it would be cutoff on the top or bottom of the screen. Also I added green output statements to the bottom display window for user validation and error messages.

In conclusion this assignment was very challenging, but I am glad to have learned and implemented the Hash Map set. I am a python programmer and can relate to this being very similar to a dictionary in python. So, I found this to be an incredible tool to add to my library. Also, I found that the use of recursion while still challenging I did get one recursive method working which helped me build a small amount of confidence in recursion. Lastly the re-visit to JavaFX was a great addition to this project and helped me build confidence with changing font colors and implementing h stacks like I have done in swift while building applications. It has been a challenging 8 weeks but I am glad to have been ahead to work on these projects and progressed through the course to build confidence putting everything I have learned in this project.

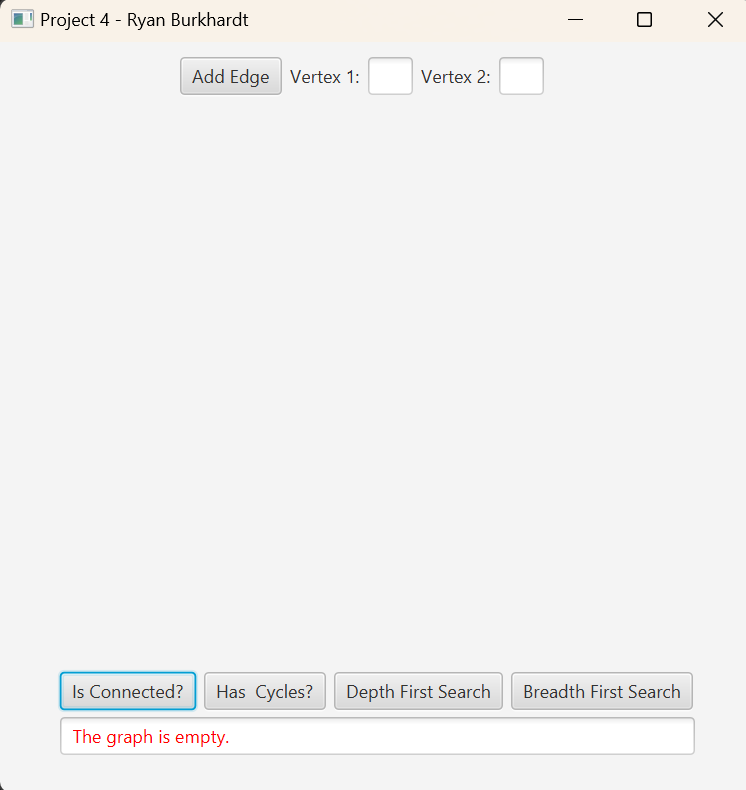
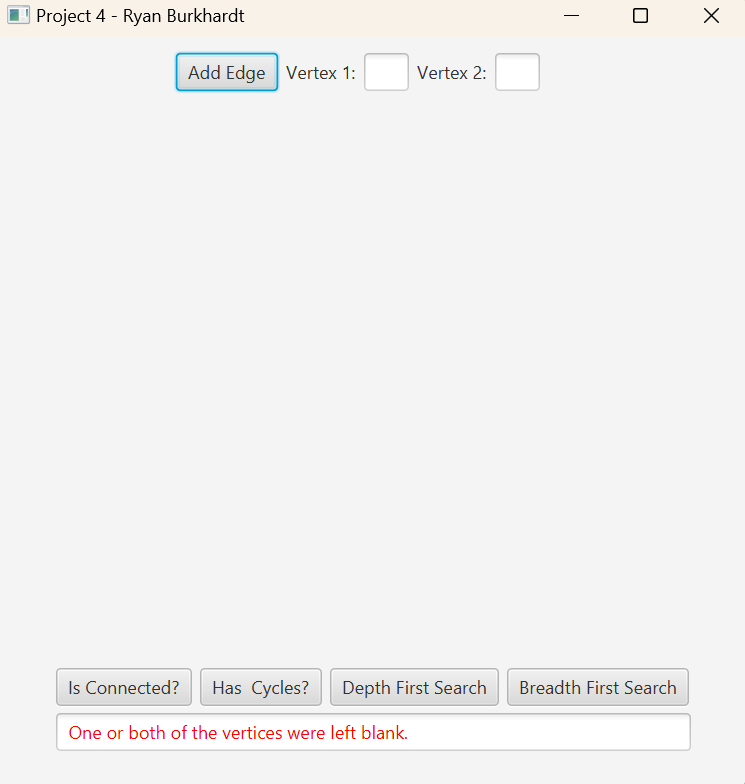
**CMSC 330 Project 1**

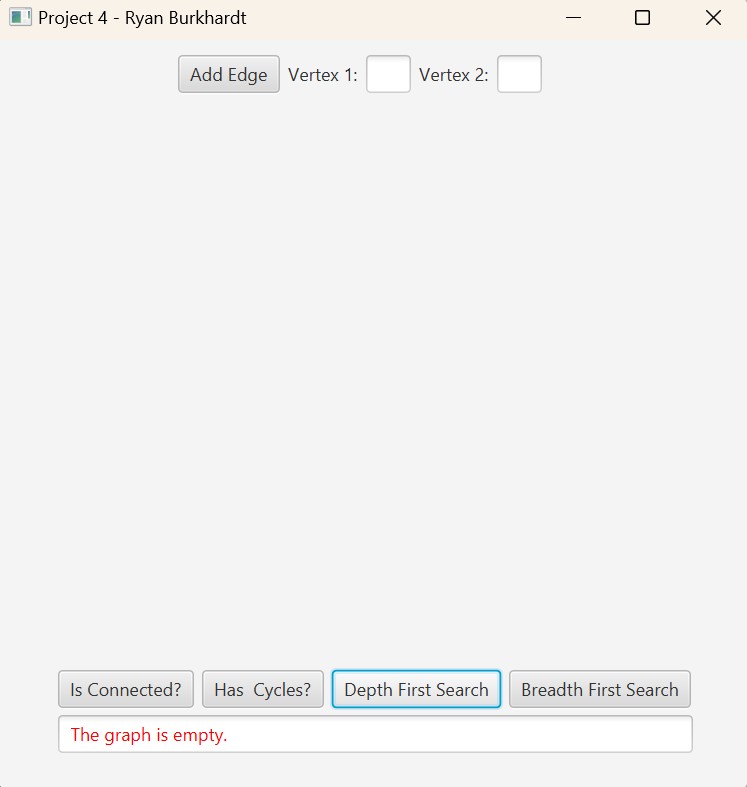
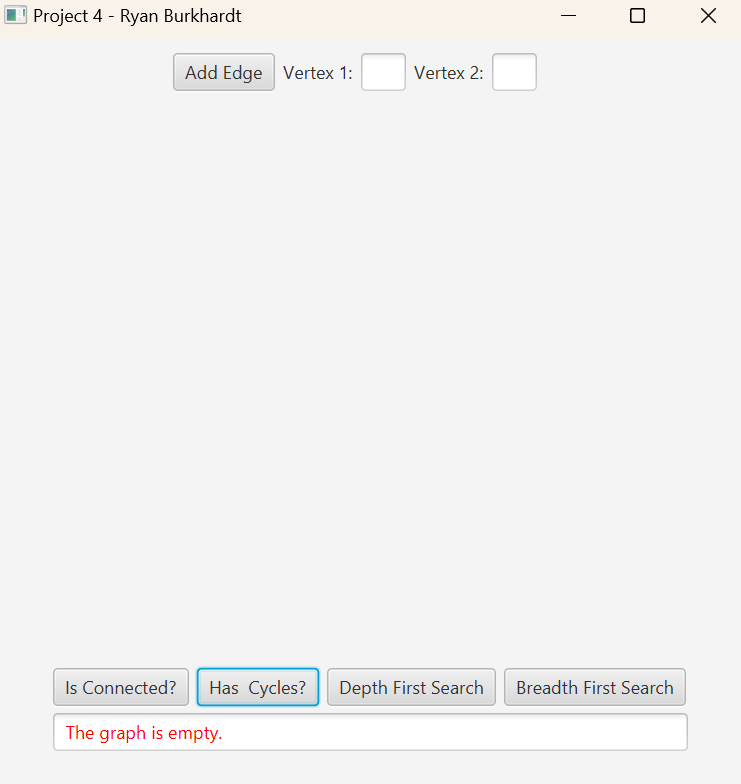
**Test Table**

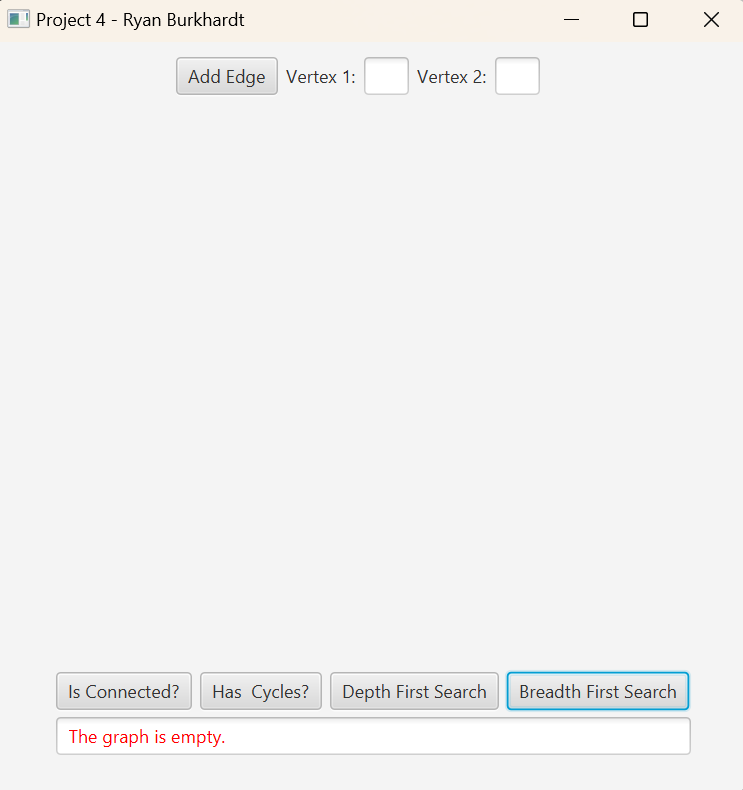
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input** | **What to Test** | **Actual Output** | **Pass/Fail** |
| 1 | Start Program()  Click all buttons:  addVertex  isConnected  checkcycle  dfssearch  bfssearch | This will test to see if JavaFX is established, and jar files are installed.  Next while leaving everything blank it will test the output text field to show every button will display graph is empty | JavaFX correctly opens the application.  Text Field displayed all appropriate messages | Pass |
| 2 | Click and add 5 vertices in a kite pattern | Properly accepts the left click and adds vertex to the application, also displays each vertex added | JavaFX correctly added all 6 vertices and displayed appropriate messages | Pass |
| 3 | Add Edges from:  A->B, A->C, B->D, C->D, D->E, E->F | Will test the add edges button as well as text fields to take in the users input and add the edges as well as displaying the outcome in the message text field | Correctly added all edges and displayed appropriate messages | Pass |
| 4 | Click isConnected  hasCycles  depth First Search  Breadth First Search | We will test to make sure all methods are correctly working and displaying in the message window | Correctly displayed all outputs. It was connected had cycles and performed depth first and breadth first search while displaying this to user. | Pass |
| 5 | Perform the exact same vertex pattern but do not create an edge from C->D  Or D->E.  Click the same buttons as test4.  Reset and only add 2 vertices.  Click the same buttons as previous. | Same test as test 4 but should have is connected false and has cycles true as well as only be able to return the vertices connected for df and bf search. Second part of test we should see that there are no edges in the graph. So appropriate message should be displayed. | The actual values were correctly displayed. The Boolean values were switched to false in appropriate cases and the dfs and bfs both only returned the values of the connected graph not the graph and a disconnected cycle or empty vertex. | Pass |

**Screenshots**

**Test Case 1**

****

****

****

**Test Case 2**

**A screenshot of a computer

Description automatically generated**

**Test Case 3**

**A screenshot of a computer

Description automatically generated**

**Test Case 4**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**Test Case 5**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**