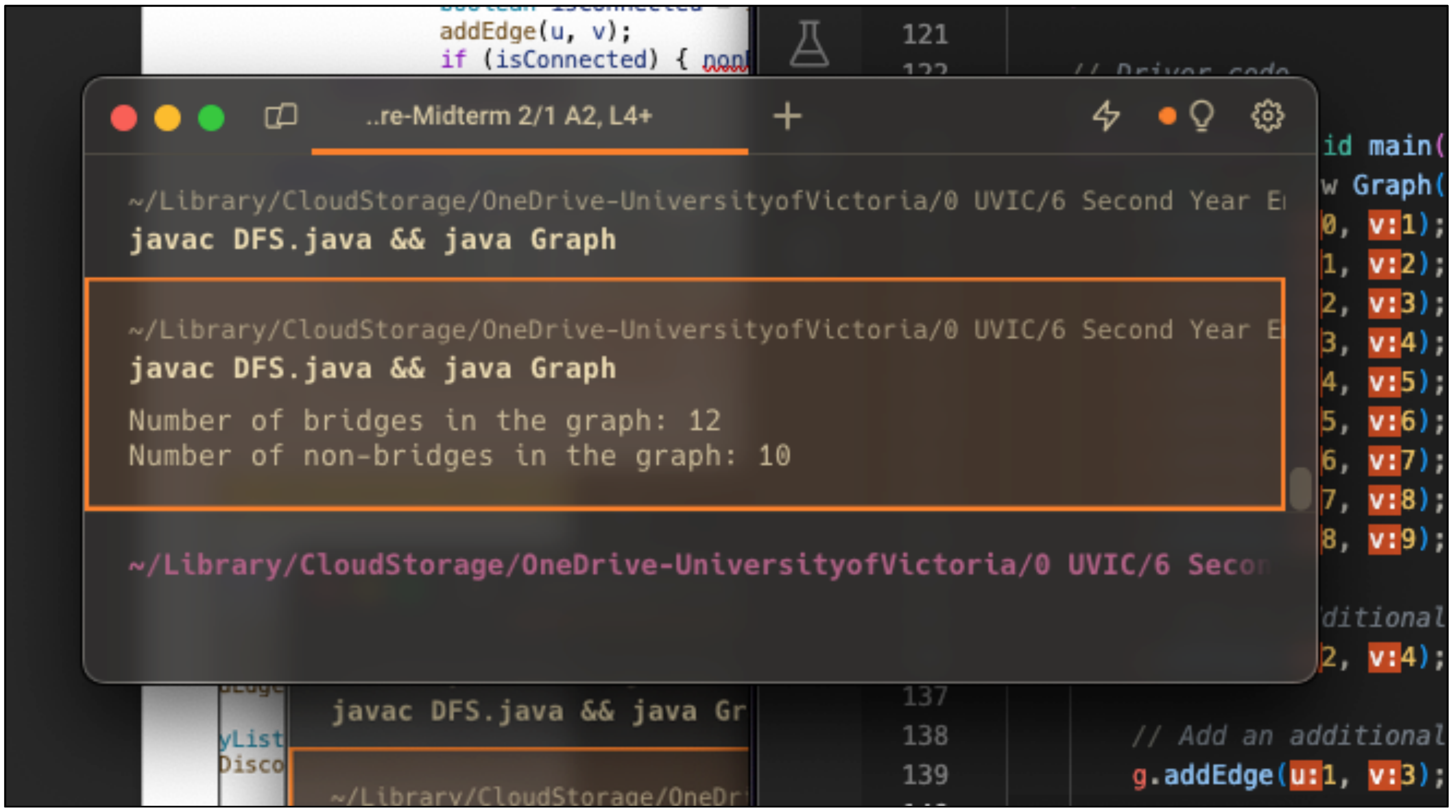
CSC 226 Assignment 2 – Question 3

```
import java.util.*;

class Graph {
    int V;
    ArrayList<ArrayList<Integer>> adj;

      private void DFS (int v, boolean[] visited) {
         visited[v] = true;
         for (Integer i : adj.get(v)) { if (!visited[i]) { DFS (i, visited); } }
      }

      public Graph (int V)
      {   this.V = V;
          adj = new ArrayList<> ();
          for (int i = 0; i < V; i++) { adj.add(new ArrayList<>()); }
      }

      public void addEdge (int u, int v) {
         adj.get(u).add(v);
         adj.get(v).add(u);
      }

      public boolean isConnected () {
         boolean[] visited = new boolean[V];
         DFS(0, visited);

        for (int i = 1; i < V; i++) { if (visited[i] == false) { return false; } }
        return true;
      }

      public int countBridges() {
         int bridgeCount = 0;
         for (int u = 0; u < V; u++) {
             List<Integer> neighbors = new ArrayList<>(adj.get(u));
             for (int v : neighbors) {
                 adj.get(u).remove(Integer.valueOf(v));
                 adj.get(v).remove(Integer.valueOf(u));
                 boolean isConnected = isConnected();
                 addEdge(u, v);
                 if (!isConnected) { bridgeCount++; } } }
         return bridgeCount;
    }

    public int countNonBridges() {
         int nonBridges = 0;
         for (int u = 0; u < V; u++) {
             List<Integer> neighbors = new ArrayList<>(adj.get(u));
             for (int v : neighbors) {
                 adj.get(u).remove(Integer.valueOf(v));
                 adj.get(v).remove(Integer.valueOf(u));
```

```
                boolean isConnected = isConnected();
                addEdge(u, v);
                if (isConnected) { nonBridges++; } } }
        return nonBridges;
    }

    public static void main(String[] args) {
        Graph g = new Graph(10);
        g.addEdge(0, 1); g.addEdge(1, 2); g.addEdge(2, 3); g.addEdge(3, 4);
        g.addEdge(4, 5); g.addEdge(5, 6); g.addEdge(6, 7); g.addEdge(7, 8);
        g.addEdge(8, 9); g.addEdge(2, 4); g.addEdge(1, 3);
        int numBridges = g.countBridges(); int nonBridges = g.countNonBridges();
        System.out.println("Number of bridges in the graph: " + numBridges);
        System.out.println("Number of non-bridges in the graph: " + nonBridges);
    }
}
```



```
~/Library/CloudStorage/OneDrive-UniversityofVictoria/0 UVIC/6 Second Year E
javac DFS.java && java Graph

~/Library/CloudStorage/OneDrive-UniversityofVictoria/0 UVIC/6 Second Year E
javac DFS.java && java Graph

Number of bridges in the graph: 12
Number of non-bridges in the graph: 10

~/Library/CloudStorage/OneDrive-UniversityofVictoria/0 UVIC/6 Secon
```

Hence, we have proved that any connected, undirected graph has a vertex whose removal, along with its incident edges, will not disconnect the graph. After adding 11 edges, we find that there are 12 bridges and 10 non-bridges. The DFS-based algorithm outlined above can be used to find such a vertex.