**NAME-YASH GANDHI. SE-IT. BATCH A**

**ROLLNO.-14**

**ISE-2 PROJECT (ADS)**

**TOPIC-GRAPH ROUTING**

**AIM:-**

To apply various route finding algorithms and display the route between the source and destination for any given graph on a GUI. And to find various features of the graph

**PROGRAMMING LANG.:-**

**JAVA (**Object Oriented Programming**).**

**Application/IDE used:-**

**NET BEANS IDE-7.1.2**

Can also be compiled and Run in

**ECLIPSE IDE** v.Photon June 2018

(Compatible for all versions)

**GUI:-use of java swing**

Swing is a GUI widget toolkit for Java.

JFrame is a class of javax.swing package extended by java.awt.frame; it adds support for JFC/SWING component architecture.

**CODE:-**Attached zip file contains the code files.

**HOW TO RUN THE CODE:-**

Save the zip file in a graphify package (folder) with extension **.java**

Open the file in Eclipse or Net Beans and run the code.

**ALGORITHMS USED:-**

* **DIJKSTRA-**

To find shortest possible path

* **BFS-**

To find shortest possible path

* **DFS-**
* **GRAPH COLORING**

**CODE OVERVIEW:-**

Two classes –Algorithms and GraphifyGUI are used.

Class Algorithms contains the code for the various algorithms used. It takes the graph, source and destination from object of class GraphifyGUI. Hashmap is used to store the distance, visit status and color of all the nodes.

Class GraphifyGUI is used to form the GUI.

For Using Jframes it extends javax.swing.jframe.

It makes an object of class Algorithms.

On clicking Start it receives the graph from the user through the text area and prints the required features on the console.

The path is displayed using dotted lines.

Randomize option is available to generate random graph and Print List is used to show the graph status.

**OUTPUT OVERVIEW-**

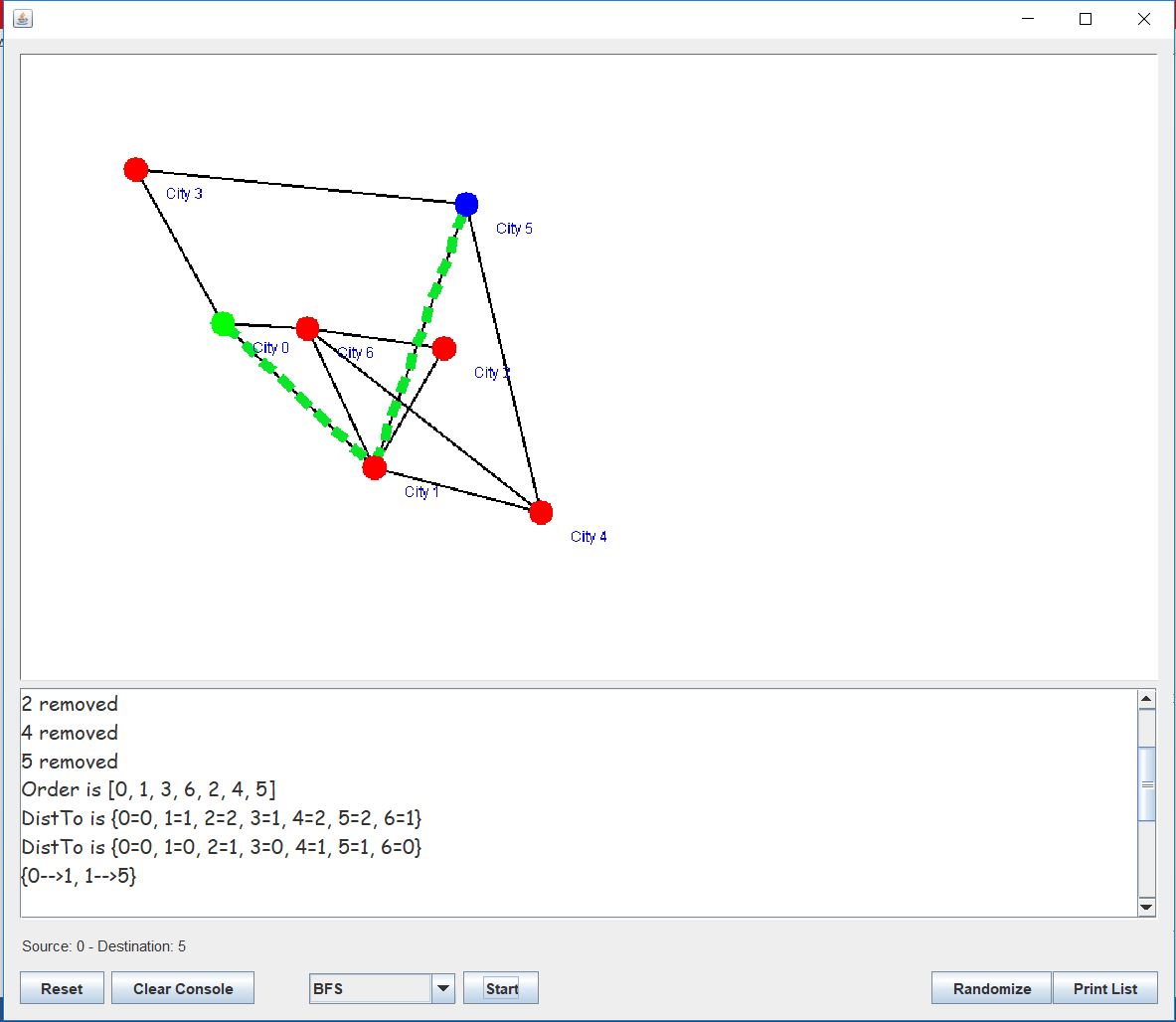
User can enter any graph structure and can find the path between any two nodes using any of the above algorithms.

The programme also assists you to find the minimum number of colors required By Graph coloring algorithm and checks if he graph is connected or not.

**Output:**

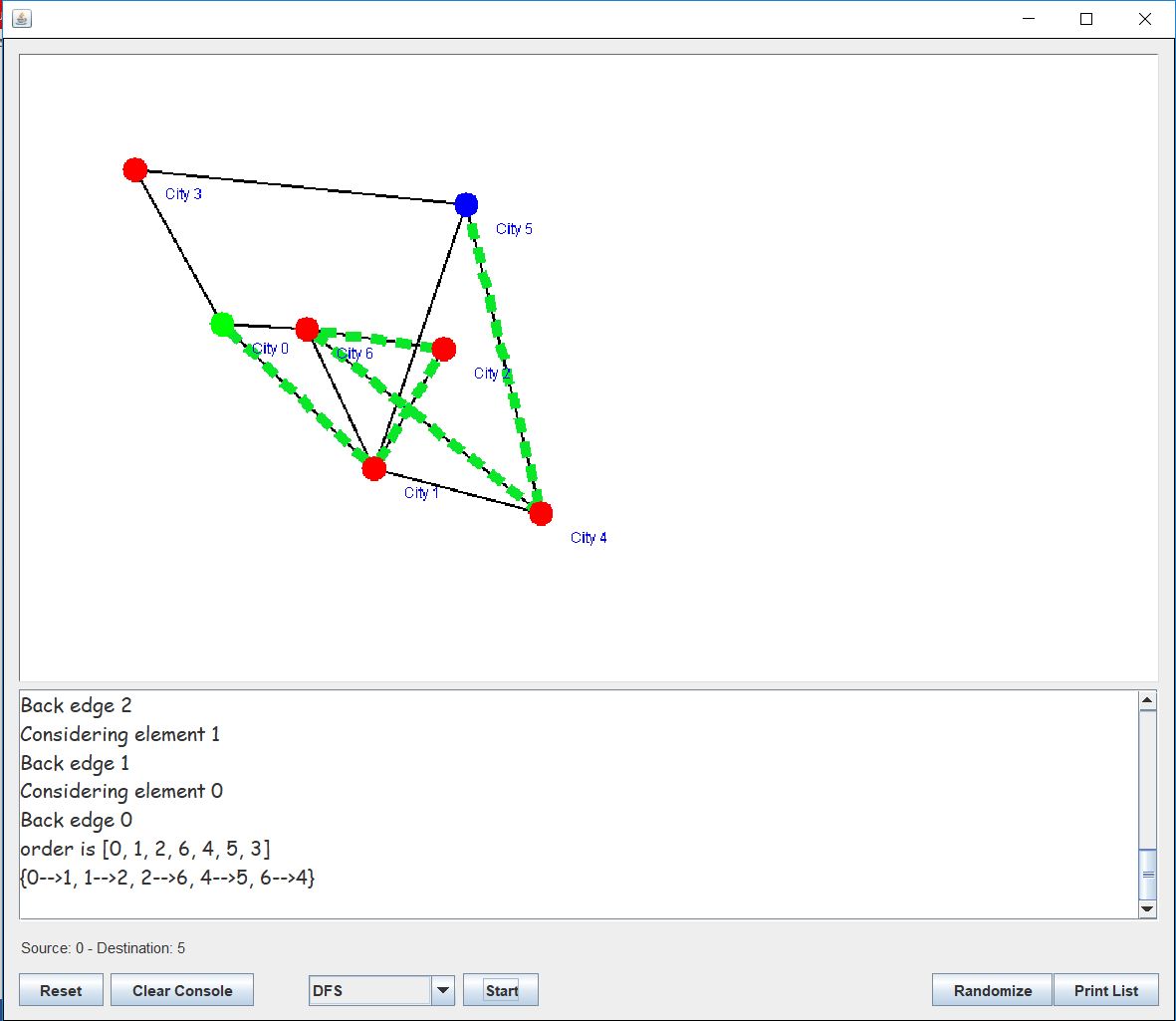
* **Breadth First Search**

Source-city 0 destination- city 5

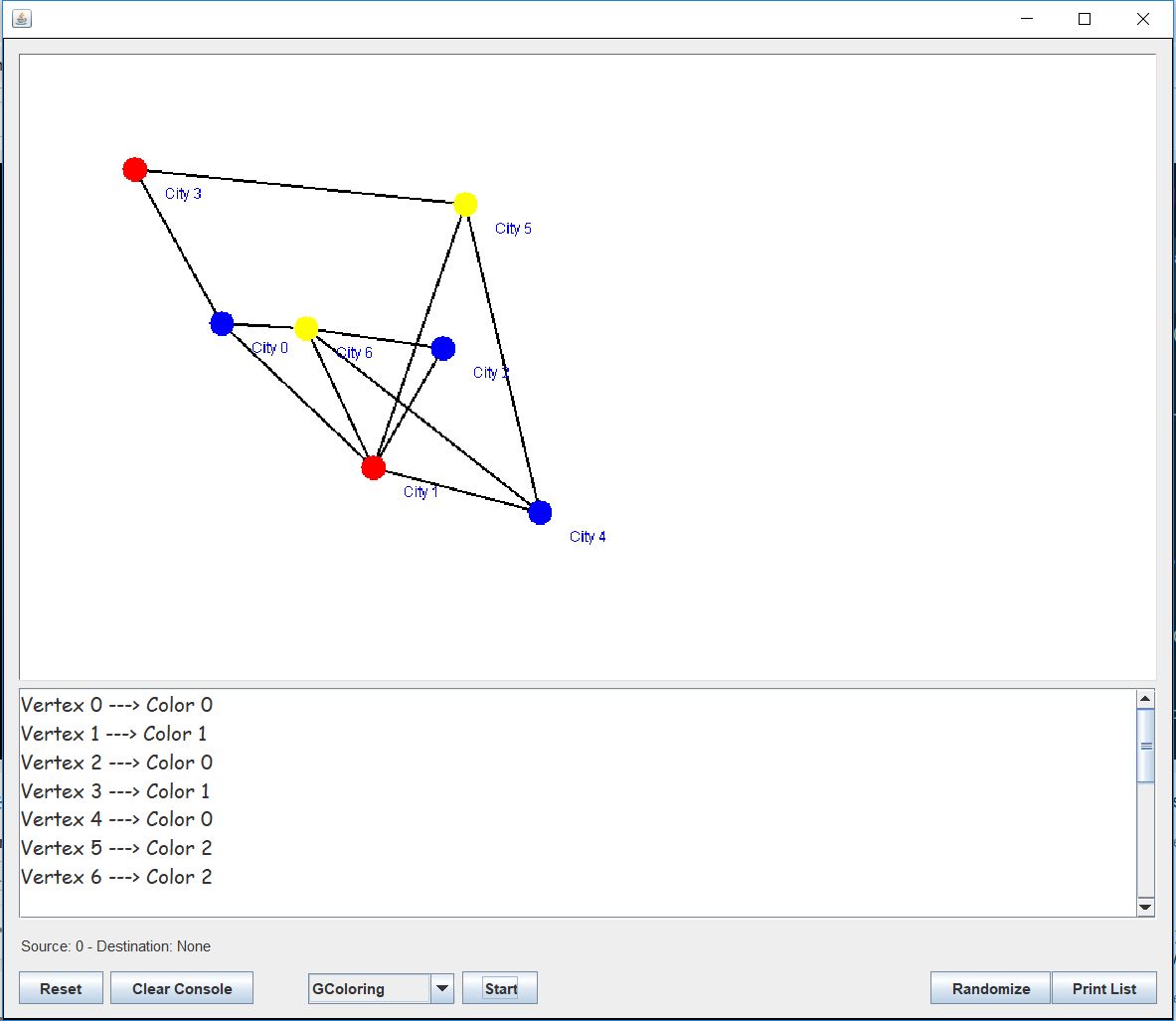


* **Depth First Search**

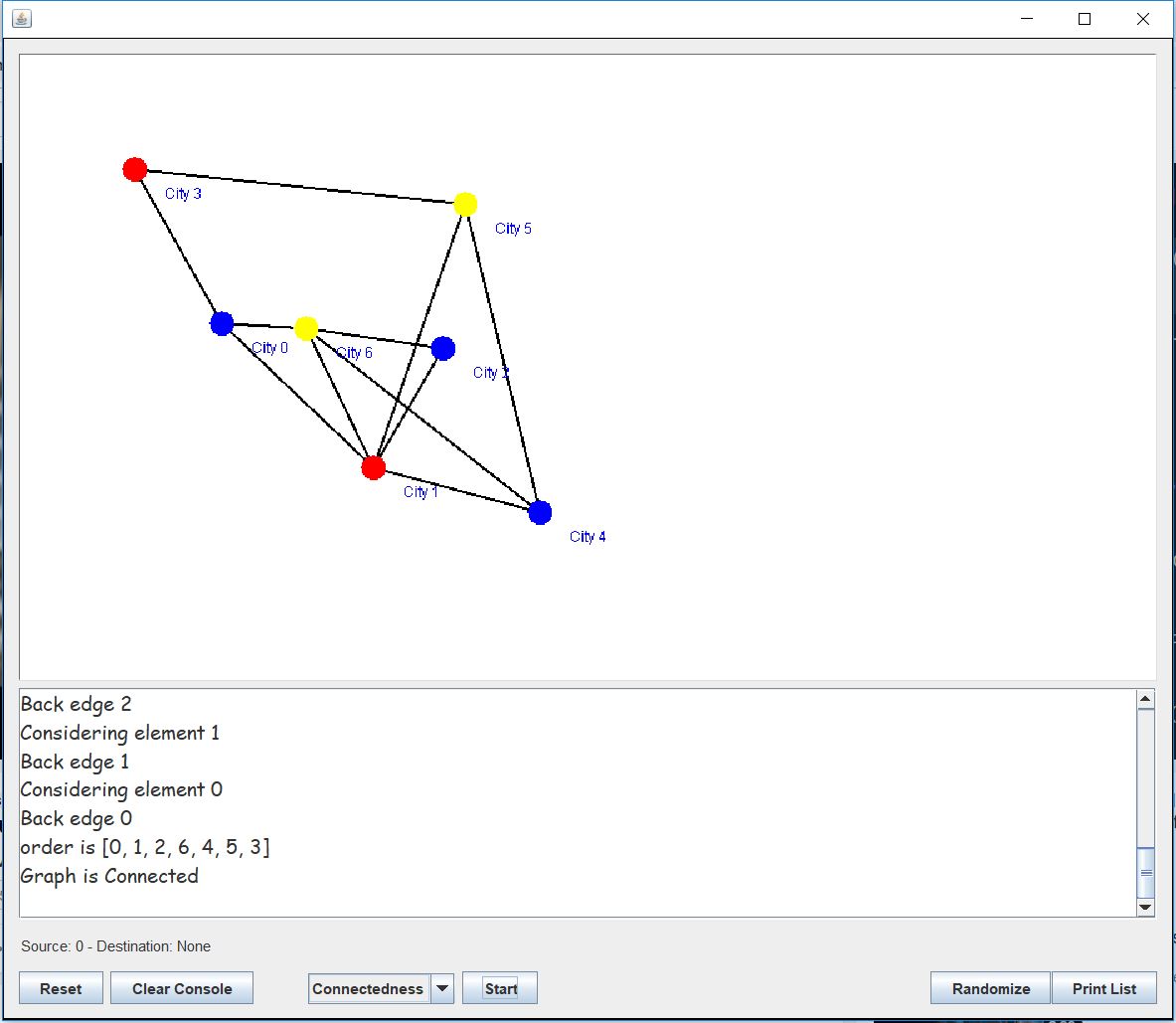
Source-city 0 destination- city 5



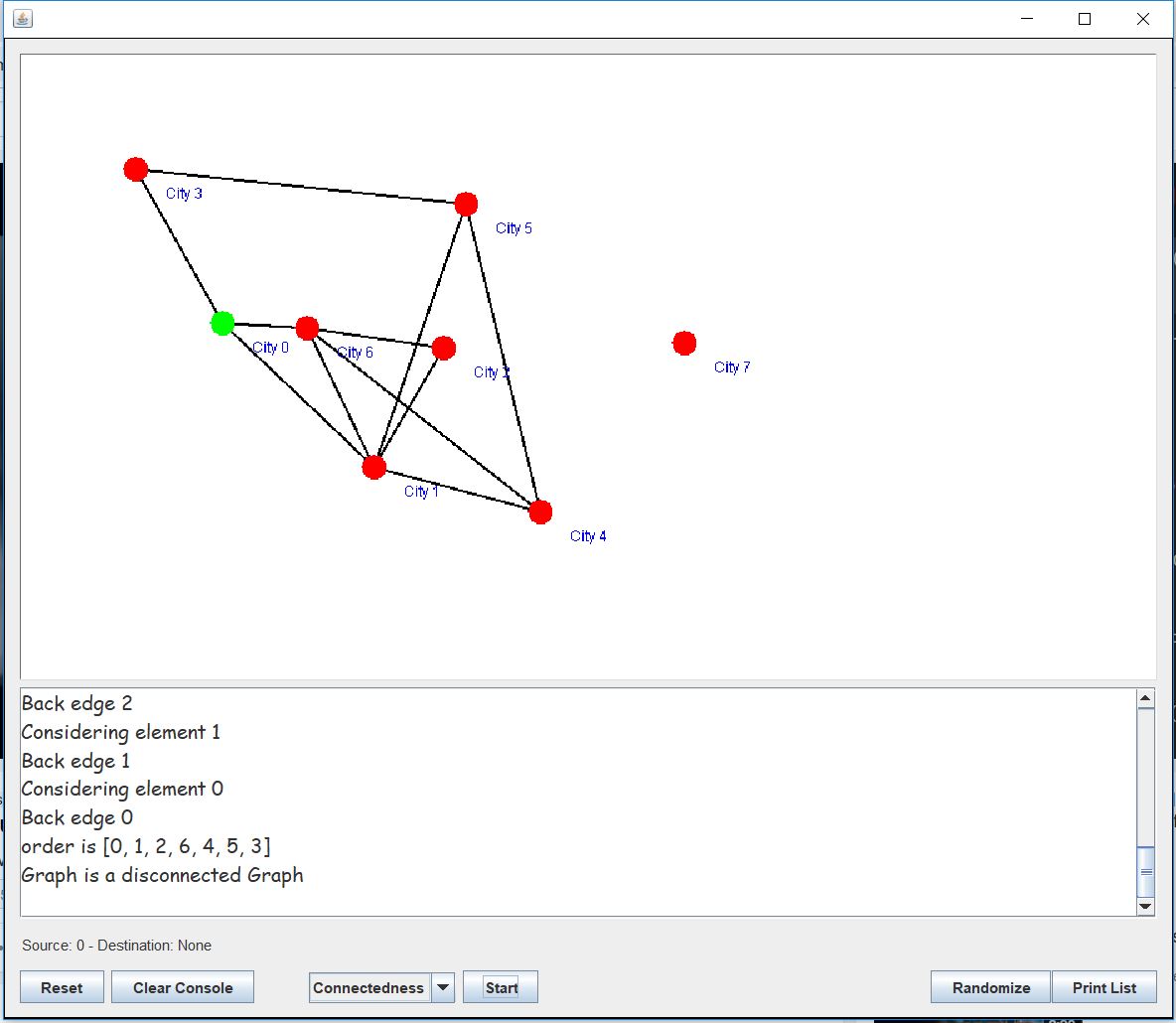
* **Greedy Colouring**



* **Graph Connected**



* **Graph not Connected**



* **Dijkstra’s Method**

Source-city 0

