

Problem Statement:

Develop an application in any language to check whether the two user given graphs are isomorphic or not. If so, convert one into another and show them graphically.

Language Used: Python

Python is a widely used high level programming language for general purpose programming, created by Guido Van Rossum and first released in 1991. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. It is a dynamic, interpreted (bytecode-compiled) language. There are no type declarations of variables, parameters, functions, or methods in source code. This makes the code short and flexible, and we lose the compile-time type checking of the source code. The language tracks the types of all values at runtime and flags code that does not make sense as it runs.

Description of various modules:

1. **Networkx:**

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. "import networkx as nx" simply imports the functions and classes from the networkx package, and the as nx part is sort of like making a nickname to access those functions easier.

Create an empty graph with no nodes and no edges.

```
>>> import networkx as nx
>>> G=nx.Graph()
```

A graph is a collection of nodes (vertices) along with identified pairs of nodes (called edges, links, etc). In NetworkX, nodes can be any hashable object e.g. a text string, an image, an XML object, another Graph, a customized node object, etc.

The graph G can be grown in several ways. NetworkX includes many graph generator functions and facilities to read and write graphs in many formats. The graph is implimented using Adjacency List.

2. **G.add_node(1):**

This adds one node at a time.

3. **G.add_edge(1,2):**

G can also be grown by adding one edge at a time.

4. **matplotlib.pyplot:**

Matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g.,

creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes. "import matplotlib.pyplot as plt" simply imports the functions and classes from the pyplot library from the matplotlib package, and the as plt part is sort of like making a nickname to access those functions easier.

5. string.ascii_uppercase:

The uppercase letters 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'. This value is not locale-dependent and will not change.

6. n1=[]:

Defining a List in Python is easy. We just need to provide name of the list and initialize it with values. Following is an example of a List in Python :

```
>>>myList = ["The", "earth", "revolves", "around", "sun"]
>>>myList
['The', 'earth', 'revolves', 'around', 'sun']
```

Here we have initialised an empty list with no elements as n1.

7. n1.append():

Append function is used to add elements such as numbers, strings, another list, dictionary etc. to a list, in this case to empty list n1.

The append method can take one or more elements as input and appends them into the list. Here is an example :

```
>>>myList.append(["a", "true"])
>>>myList
['The', 'earth', 'revolves', 'around', 'sun', ['a', 'true']]
```

So, the two values were appended towards the end of the list. Whether we add one element or multiple elements to the list, if we have used append then they will be added as a single element only. The append() method only modifies the original list. It doesn't return any value.

8. G.degree():

The degree of a vertex v in a graph is the number of edges connecting it, with loops counted twice. The degree of a vertex v is denoted $\deg(v)$. The maximum degree of a graph G , denoted by $\Delta(G)$, and the minimum degree of a graph, denoted by $\delta(G)$, are the maximum and minimum degree of its vertices.

If all the degrees in a graph are the same, the graph is a regular graph.

This function returns the degree of a node or nodes.

9. l1.sort():

The sort() method sorts the elements of a given list in a specific order - Ascending or Descending. By default, sort() doesn't require any extra parameters. Sort() method doesn't return any value. Rather, it changes the original list.

10. plt.rcParams['figure.facecolor'] = 'salmon':

The default interactive figure background will be changed to salmon colour.

11. plt.plot(linewidth=3):

This changes the thickness/width of the line samples featured in the pyplot legend and sets it to 3.

12. `mng = plt.get_current_fig_manager()`

`mng.window.state('zoomed')`:

It creates a maximized window that snaps to the edges of the screen and has the minimize, maximize/restore down and close buttons as it should.

13. `pos = nx.random_layout(G)`:

This command position nodes uniformly at random in the unit square.

For every node, a position is generated by choosing each of dim coordinates uniformly at random on the interval [0.0, 1.0). It returns a dictionary of positions keyed by node.

14. `nx.draw_networkx(G,pos)`:

This command draws the graph G using Matplotlib options for node positions, labeling, titles, and many other drawing features.

15. `plt.title()`:

Mathplotlib displays the plot title passed to it as a parameter within parenthesis in the centre.

16. `plt.show()`:

This will produce an interactive plot on the screen and displays the current figure that we are working on, assuming we are using a backend (renderer) that supports plotting to our user interface.

17. `plt.rcParams['figure.facecolor'] = 'teal'`:

The default interactive figure background will be changed to teal colour.

18. `plt.rcParams['figure.facecolor'] = 'green'`:

The default interactive figure background will be changed to green colour.

19. `diction = {x:i for x, i in enumerate(al, 1)}`:

`enumerate()` is one of the built-in Python functions. It returns an enumerate object.

`enumerate(iterable, start=0)`-

Return an enumerate object. “iterable” must be a sequence, an iterator, or some other object which supports iteration. The `__next__()` method of the iterator returned by `enumerate()` returns a tuple containing a count (from start which defaults to 0) and the values obtained from iterating over iterable. Here the “diction” acts as a mapping dictionary to store all the uppercase alphabets with which the graph will be later relabelled.

20. `nx.relabel_nodes`:

This relabels the nodes of the graph G.

`networkx.relabel.relabel_nodes(G, mapping, copy=True)` -

The parameter G is a Graph, the mapping has to be a dictionary and the last parameter is optional. If copy is set to True, - which is the default - a copy will be returned, otherwise, i.e. if it is set to False, the nodes of the graph will be relabelled in place.

ALGORITHM

The nodes of first graph have been taken numerically and that of the second graph have been taken alphabetically. In the output the graph with nodes named numerically is converted into graph with nodes named alphabetically. The user has to give the input in the format as shown below and the output will be displayed accordingly.

Input Format:

The first line of input contains the number of nodes in graph 1.

The second line of input contains the number of edges(N).

The user has to then provide N lines of input,each of which contains 2 space separated INTEGERS which denote the nodes containing the edge.

The next line of input asks for the number of nodes in graph 2.

Enter the number of edges for the second graph(M).

The user has to then provide M lines of input,each of which contains 2 space separated ALPHABETS which denote the nodes containing the edge.

Output Format:

A window will appear representing the graph 1 and graph 2 graphically one after the other.

If isomorphic,a third window will pop up showing the transformation from graph 1 to graph 2 and the message "Graphs are isomorphic" is printed on the input screen and if they are not, "Graphs are not isomorphic" is printed on the screen.

Sample Input 1:

Enter no. of Nodes in Graph 1: 4

Enter no. of edges in graph 1: 6

1 2

1 3

1 4

2 4

2 3

3 4

Enter no. of Nodes in Graph 2: 4

Enter no. of edges in graph 2: 6

A B

A C

A D

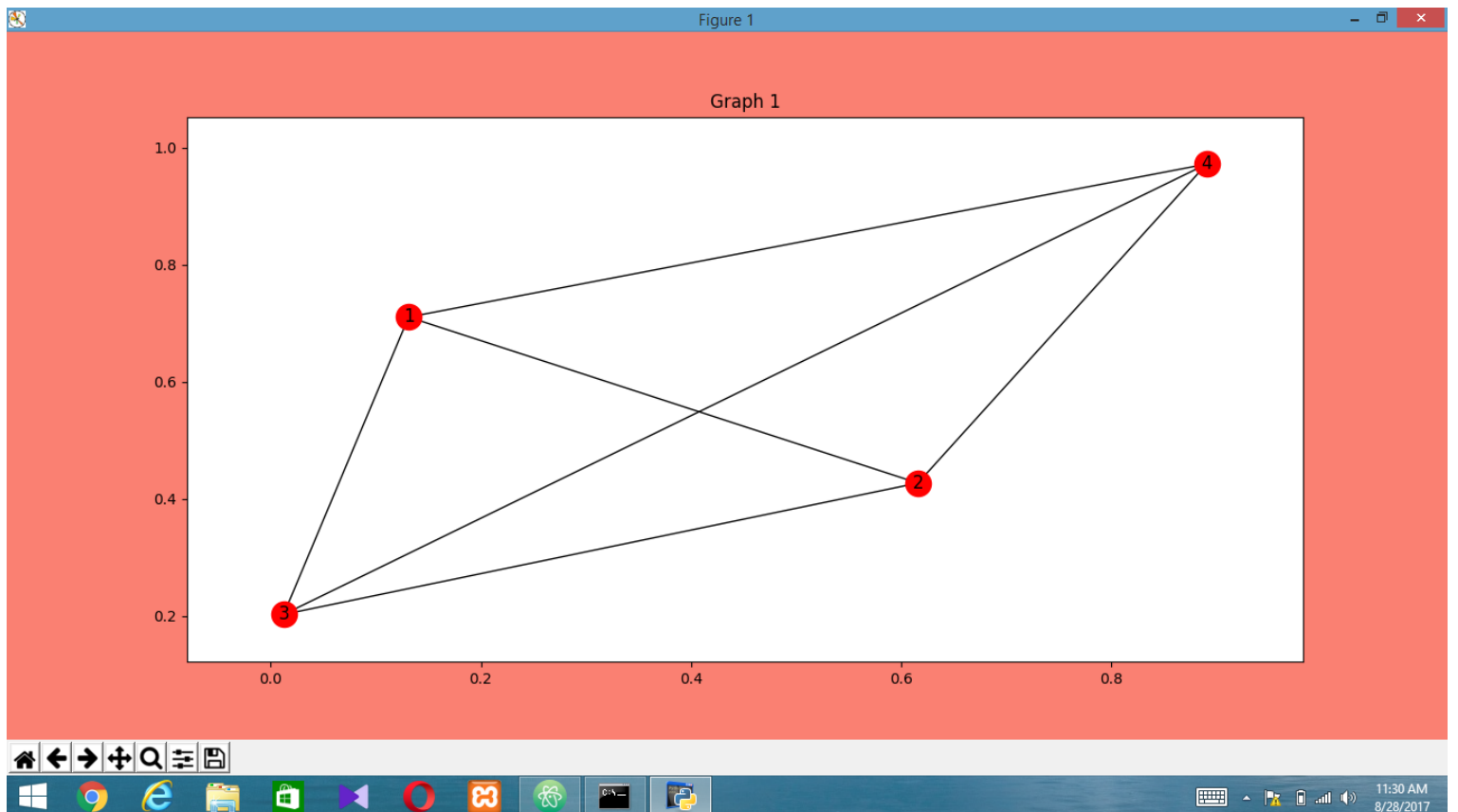
B C

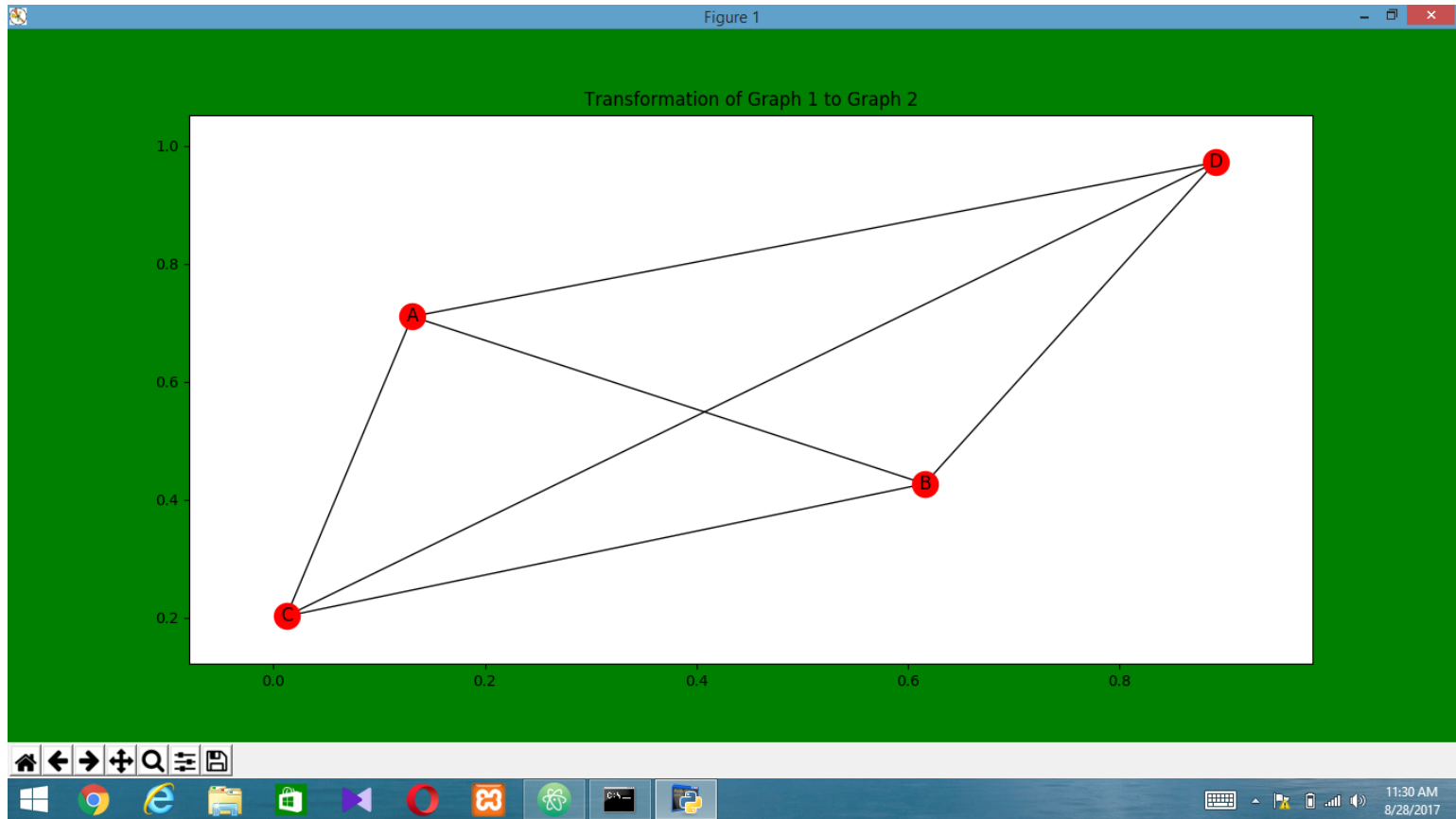
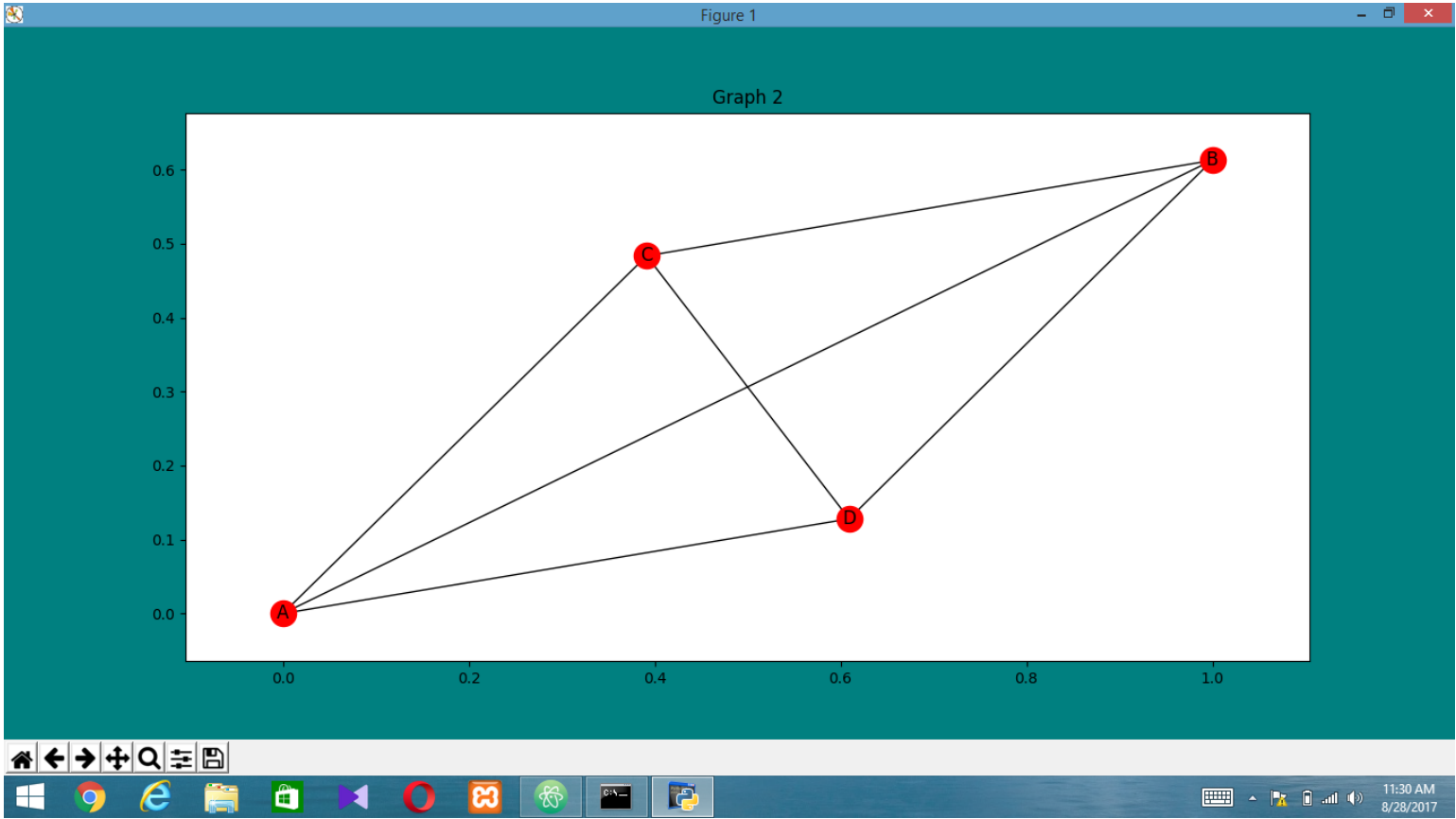
B D

C D

Sample Output 1:

Graphs are isomorphic.





```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Shreya>python mycode.py
Enter no. of Nodes in Graph 1: 4
Enter no. of edges in graph 1: 6
1 2
1 3
1 4
2 4
2 3
3 4
Enter no. of Nodes in Graph 2: 4
Enter no. of edges in graph 2: 6
A B
A C
A D
B C
B D
C D
Graphs are isomorphic
```

Sample Input 2:

Enter no. of Nodes in Graph 1: 3

Enter no. of edges in graph 1: 3

1 2

1 3

2 3

Enter no. of Nodes in Graph 2: 4

Enter no. of edges in graph 2: 6

A B

A C

A D

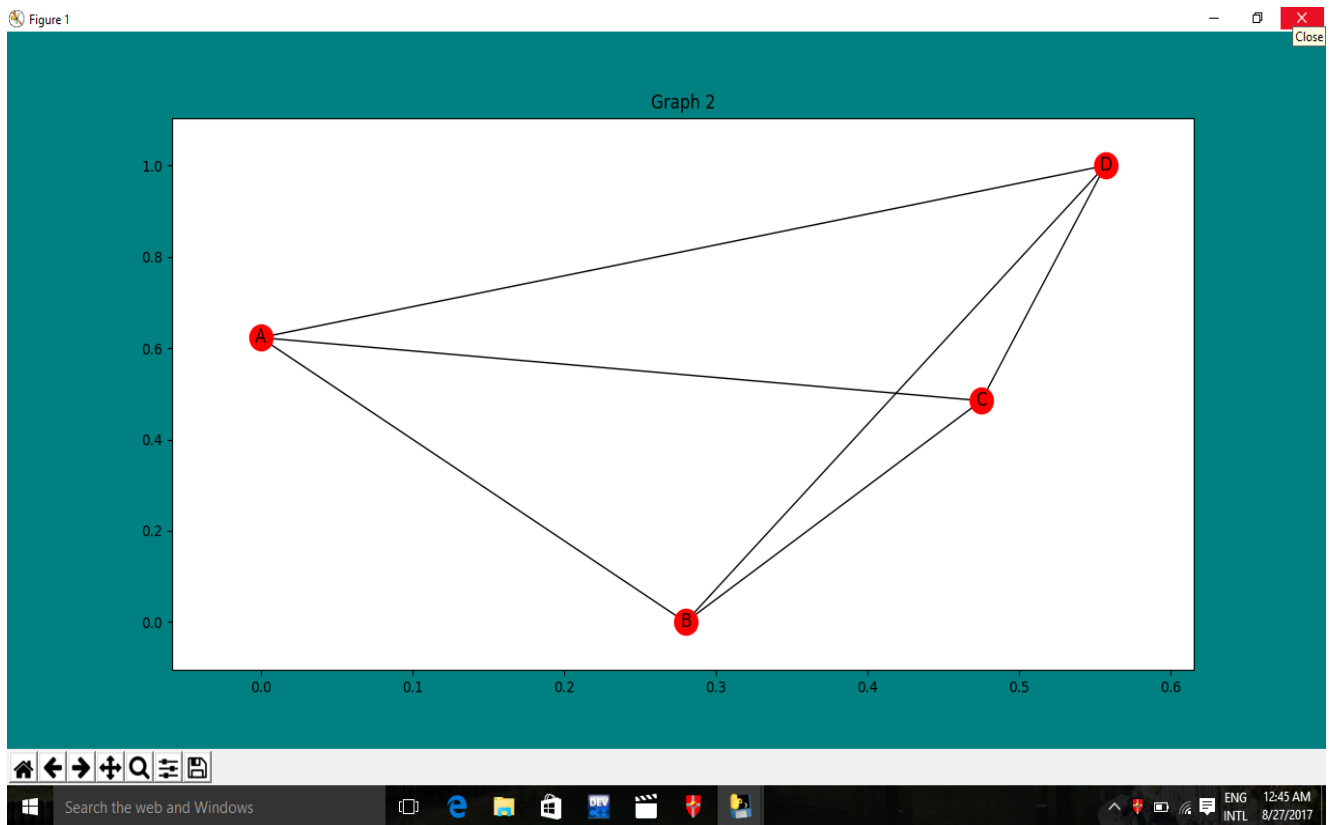
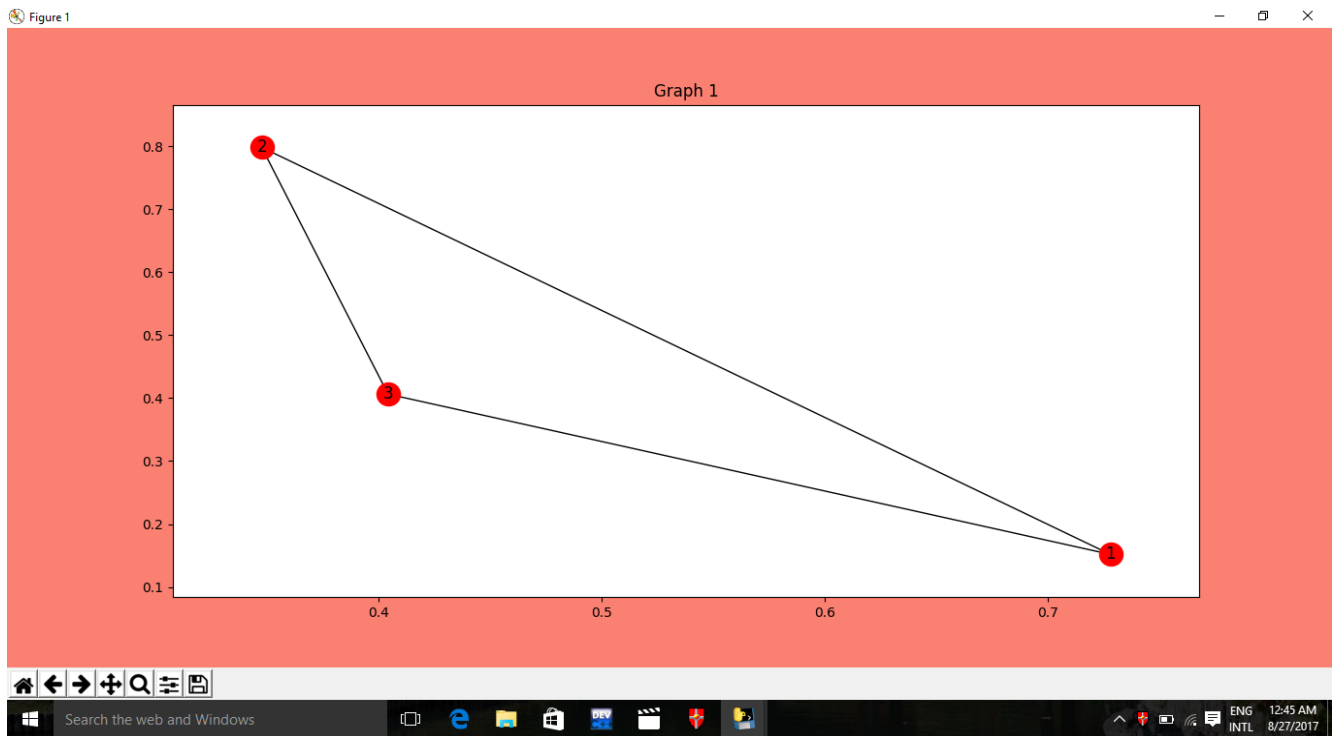
B C

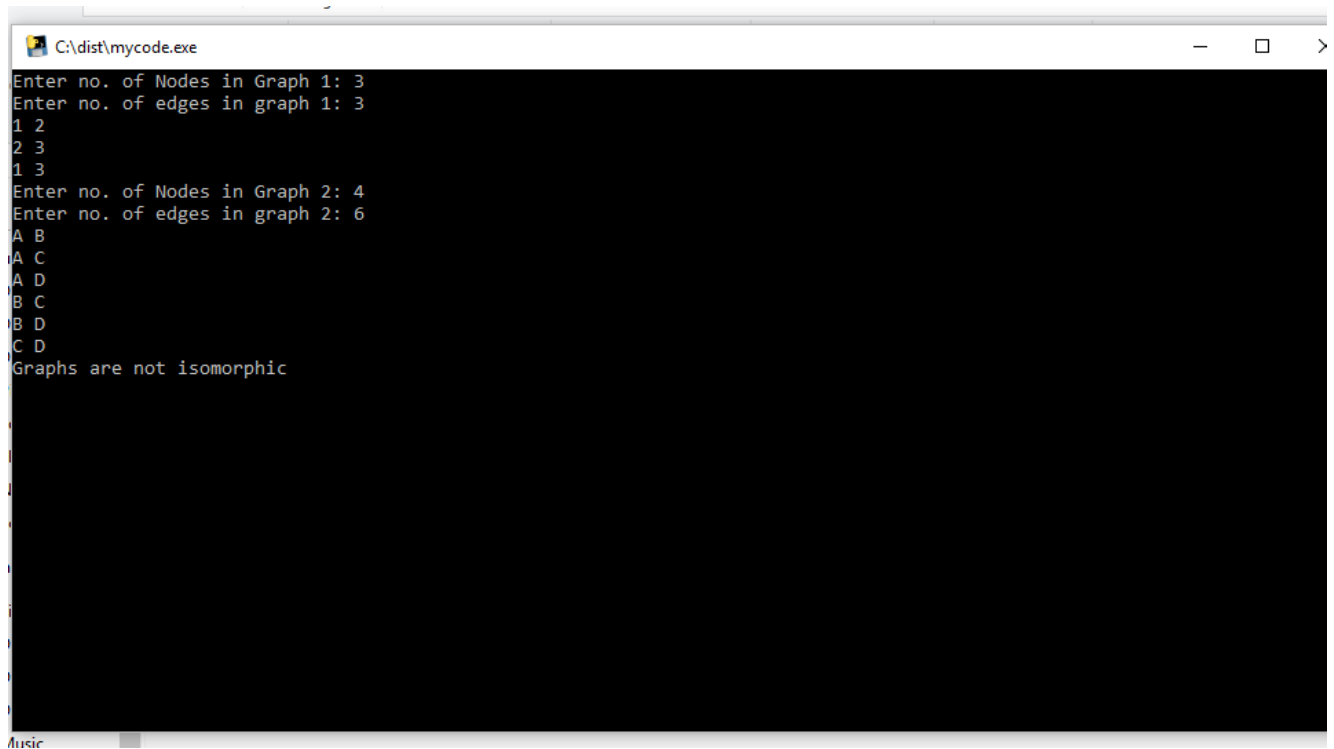
B D

C D

Sample Output 2:

Graphs are not isomorphic.





```
C:\dist\mycode.exe
Enter no. of Nodes in Graph 1: 3
Enter no. of edges in graph 1: 3
1 2
2 3
1 3
Enter no. of Nodes in Graph 2: 4
Enter no. of edges in graph 2: 6
A B
A C
A D
B C
B D
C D
Graphs are not isomorphic
```

Limitations:

- The self-loops and parallel edges cannot be drawn graphically, though the code handles the same very smoothly.
- The nodes of the graph are not movable. This is because matplotlib (the package for drawing graph) draws graph randomly.