RAMAIAH UNIVERSITY
OF APPLIED SCIENCES

ASSIGNMENT

Course Code	19CSC311A
Course Name	Graph Theory and Optimization
Programme	B.Tech.
Department	Computer Science and Engineering
Faculty	Faculty of Engineering and Technology

Name of the Student	Subhendu Maji
Reg. No	18ETCS002121
Semester/Year	6TH / 2018
Course Leader/s	

Declaration Sheet			
Student Name	Subhendu Maji		
Reg. No	18ETCS002121		
Programme	B.Tech.	Semester/Year	6th /2018
Course Code	19CSC311A		
Course Title	Graph Theory and Optimization		
Course Date		to	
Course Leader			

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Course Leader and date	Signature of the Reviewer and date

Contents

Assignment-1			
Reg. No.	18ETCS002121	Name of Student	Subhendu Maji

Sections	Marking Scheme		Marks		
			Max Marks	First Examiner Marks	Moderator
Part A					
	A.1.1	Answers and Justification	06		
	A.1.2	Algorithm	04		
	A.1.3	Code and results	06		
		Part-A Max Marks	16		
Part B					
	B.1.1	Detailed Explanation of Approach	04		
	B.1.2	Algorithm	05		
		Part-B Max Marks	09		
		Total Assignment Marks	25		

Course Marks Tabulation				
Component-1 (B) Assignment	**First Examiner**	**Remarks**	**Moderator**	**Remarks**
A				
B				
Marks (out of 25)				

Signature of First Examiner

Signature of Moderator

Solution to Question No. 1:

The **Seven Bridges of Königsberg** is a historically notable problem in mathematics. Its negative resolution by Leonhard Euler in 1736 laid the foundations of graph theory and prefigured the idea of topology.

The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands—Kneiphof and Lomse—which were connected to each other, or to the two mainland portions of the city, by seven bridges. The problem was to devise a walk through the city that would cross each of those bridges once and only once.

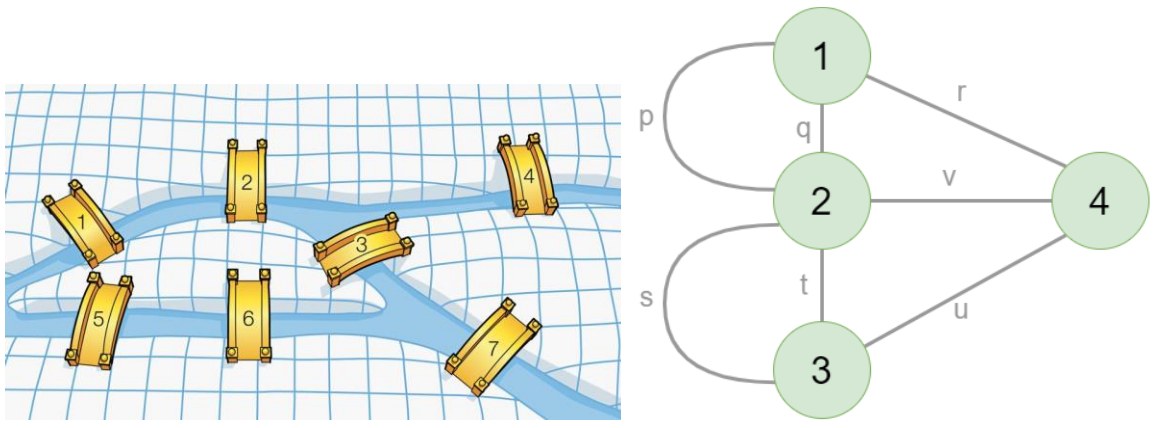


Figure 1 Königsberg Graph

Graph $G(V, E)$
Land Masses, Vertices $V = [1,2,3,4]$
Bridges, Edges $E = [p, q, r, s, t, u, v]$
There are 4 land masses and 7 Bridges. So, our graph will have 4 vertices and 7 edges.

The number of vertices of odd degree must be either zero or two. If not, then there is no "Euler Path”. And if there are two vertices with odd degree, then they are the starting and ending vertices.

A.1.1 Answers and Justification

a. **Can the people of Königsberg successfully walk over all the bridges once and get back to where they started?**

   No, it is impossible to walk over all the bridges only once and get back to where they started.

   Justification

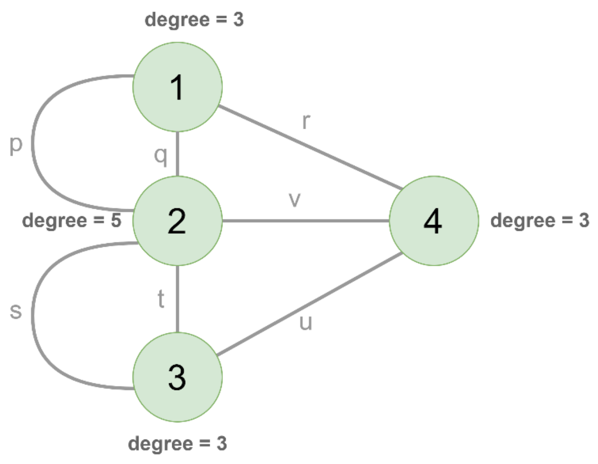


Figure 1 königsberg with degree

the question can be interpreted as whether or not there exists a Euler Circuit (traverse through each edge only once and reach back at the starting vertex).

We know, any connected graph, in which all edges are traversed at least once and start and end at the same vertex is called and **Euler Circuit**.

Any graph to be Euler circuit, **every vertex must have Even degree**.

As in Figure 1 we can observe that, Vertices 1, 4 and 3 have degree = 3 and Vertex 2 has degree = 5,

This graph has four vertices of odd degree. So, it does not have a Euler Path or Euler Circuit.

No landmass is attached to an even number of bridges. Hence, there is no round trip walk.

b. Can the bridge walk be achieved if the people were happy not returning to their starting point?

No, even if the people were happy not returning to their starting point, all the bridges cannot be visited at least once i.e. It is not possible to make a Euler path.
Since in our graph, all the vertices have odd degree, it is not possible to find a Euler path.

Justification

For a graph to be Euler graph, vertices must not have odd degrees at all or only two vertices with odd degree. The two vertices with odd degree will be start and end vertex.

In the Königsberg problem, there does not exist a Euler Path as the criteria for a graph to have a Euler Path is that there can only be either zero number of odd degrees or two number of odd degrees.

As it is show in Figure 1, four vertices have an odd degree and thus a Euler Path cannot exist for the given graph.
Therefore, it is not possible for the people of Königsberg to successfully walk over all the bridges once without caring about returning to the same place.

c. **If one or more of the bridges were removed, can the round trip walk around the bridges of Königsberg be achieved?**

Yes, by removing 2 bridges we can make a round trip walk around the bridges. i.e., if 7 bridges are reduced to 5 bridges then a round trip/ Euler circuit is possible.

Justification

In order to make the round trip (Euler Circuit), we need all vertices to have an even degree, thus implying that we have to remove at the least two edges, which would make all vertices have an even degree thus ensuring a Euler Circuit. While removing the two edges, any edge can be the first edge removed, however, to ensure that all vertices have and even degree, the second edge removed has to be one of the edges between the nodes not connected to the first edge removed, as seen in Figure 2.

By removing vertices r and t, we get graph which is shown in Figure 2.

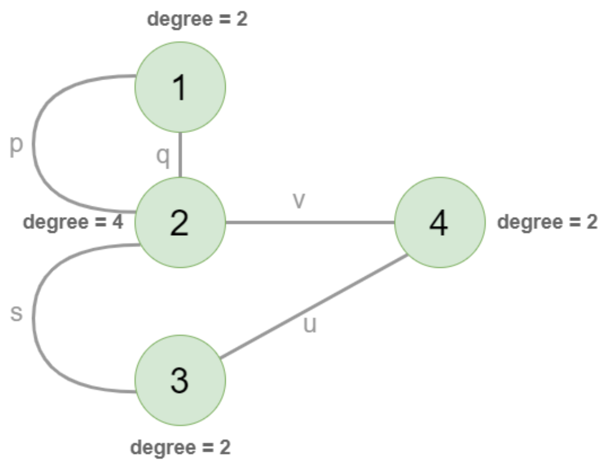


Figure 2 Walk Possible with 5 bridges

Then the round walk (Euler circuit) can be

$$1 \rightarrow p \rightarrow 2 \rightarrow s \rightarrow 3 \rightarrow u \rightarrow 4 \rightarrow v \rightarrow 2 \rightarrow q \rightarrow 1$$

d. If the city of Königsberg had seven bridges arranged in some other way, will it be possible to make the round trip walk successfully?

Yes, it is possible to arrange the seven bridges in such a way that a round trip walk can be made successfully. This is achieved by making it so that every vertex in the graph has an even degree as seen in Figure 3.

Justification

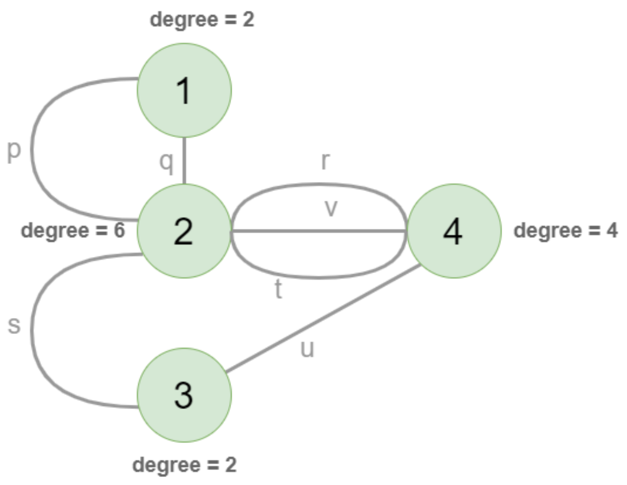


Figure 3 Rearranging seven bridges

We can consider any vertex as start since the degree of all the vertices are even.

And the path (Euler Circuit) will be

$$\mathbf{2 \rightarrow \boldsymbol{s} \rightarrow 3 \rightarrow \boldsymbol{u} \rightarrow 4 \rightarrow \boldsymbol{t} \rightarrow 2 \rightarrow \boldsymbol{v} \rightarrow 4 \rightarrow \boldsymbol{r} \rightarrow 2 \rightarrow \boldsymbol{q} \rightarrow 1 \rightarrow \boldsymbol{p} \rightarrow 2}$$

e. If the seven bridges of Königsberg are replaced with eight, nine and ten bridges, what can be commented about the land masses in each of the cases?

Case 1: With 8 bridges

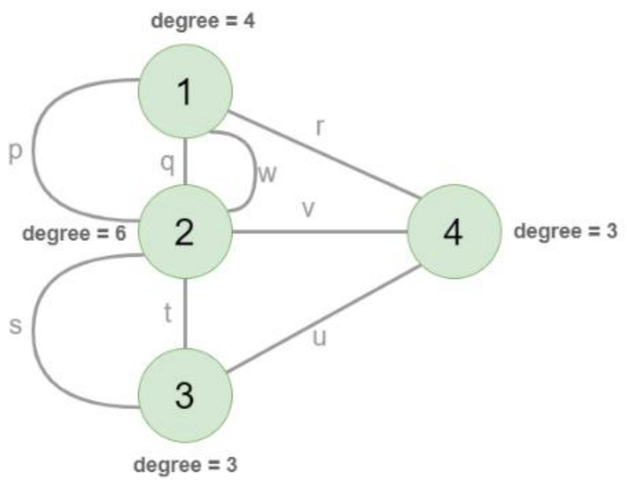


Figure 4 new edge w is added to graph

As in Figure 4 new edge w is added. It is not possible to traverse every edge once and reach to its starting point as 2 vertices have odd degree.

Since 2 vertices has odd degree, we can make them as start and end vertices. Then Euler Path is possible.

Euler circuit is not possible, but Euler's path is possible. Start at different vertex and end at different vertex.

Euler Path: $3 \rightarrow s \rightarrow 2 \rightarrow v \rightarrow 4 \rightarrow u \rightarrow 3 \rightarrow t \rightarrow 2 \rightarrow p \rightarrow 1 \rightarrow w \rightarrow 2 \rightarrow q \rightarrow 1 \rightarrow r \rightarrow 4$

Justification

we can also make every vertex have an even degree and thus ensure that there exists a Euler Circuit where eight bridges are used between four land masses. Therefore, we can travel through every land mass by crossing every bridge exactly once, <u>however this solely depends on how we arrange the bridges so as to achieve even degrees for every vertex, it is not guaranteed for every arrangement of bridges</u>.

e.g.

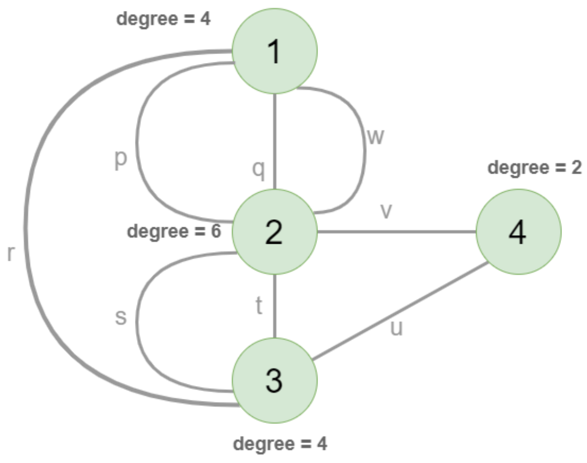


Figure 5 Eight Bridges

As it can be seen in Figure 5,

Euler Circuit: $1 \rightarrow p \rightarrow 2 \rightarrow s \rightarrow 3 \rightarrow t \rightarrow 2 \rightarrow v \rightarrow 4 \rightarrow u \rightarrow 3 \rightarrow r \rightarrow 1 \rightarrow q \rightarrow 2 \rightarrow w \rightarrow 1$

Case 2: With 9 bridges

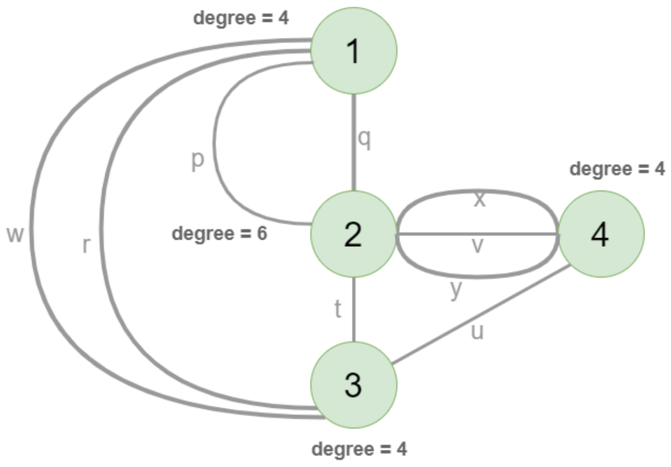


Figure 6 Nine bridges

As it can be seen in Figure 6, by having nine bridges,

Euler circuit:

$$4 \rightarrow u \rightarrow 3 \rightarrow w \rightarrow 1 \rightarrow r \rightarrow 3 \rightarrow t \rightarrow 2 \rightarrow p \rightarrow 1 \rightarrow q \rightarrow 2 \rightarrow v \rightarrow 4 \rightarrow x \rightarrow 2 \rightarrow y \rightarrow 4$$

Justification

we can make every vertex have an even degree and thus ensure that there exists a Euler Circuit where nine bridges are used between four land masses. Therefore, we can travel through every land mass by crossing every bridge exactly once, however this solely depends on how we arrange the bridges so as to achieve even degrees for every vertex, it is not guaranteed for every arrangement of bridges.

Case 3: With 10 bridges

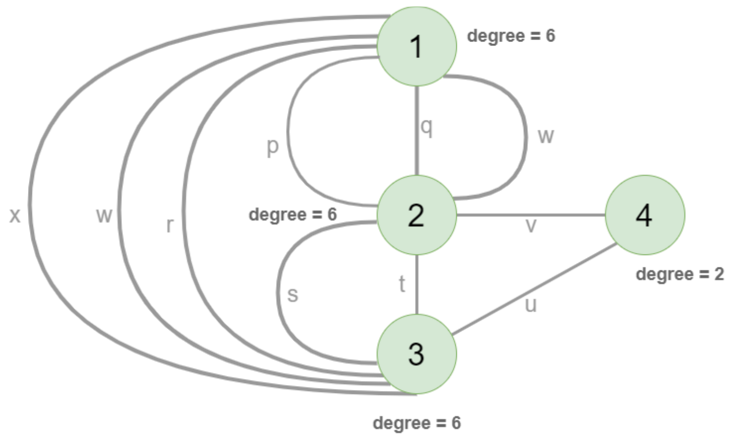


Figure 7 Ten Bridges

As it can be seen in Figure 7, by having ten bridges, we can make every vertex have an even degree and thus ensure that there exists a Euler Circuit where ten bridges are used between four land masses. Therefore, we can travel through every land mass by crossing every bridge exactly once, however this solely depends on how we arrange the bridges so as to achieve even degrees for every vertex, it is not guaranteed for every arrangement of bridges.

f. To cover n number of bridges is there a generalized result? Is there any relevance of knowing in what way the bridges are connected to the land masses?

In order to cover n bridges exactly once and ensure that we reach back to the same starting point, we need to ensure that each land mass (vertex) has only an even number of bridges attached to that land mass (all vertices need to have even degree to ensure Euler Cycle).

However, if we were to cover n bridges exactly once without having the condition of reaching back to the same starting point (Euler Path), we need to ensure that there are either zero or exactly two land masses (vertices) which have an odd degree.
Thus, knowing in what way the bridges are connected to the land masses is of utmost importance as there existing a Euler Path/Circuit all depends on the degree of each land mass.

A.1.2 Algorithm

We are given p number of land masses and q number of bridges. We have to find all the possible path through every land mass using every bridge only once (if possible).

If we visualize this problem in graphs, the land masses can be vertices and bridges can be the edge.

```
STEP 1: Start
STEP 2: Check if it is a Eulerian Graph
      2.1 if neither 0 odd degree vertices nor 2 odd degree vertices
            2.1.1 then, “NOT A EULERIAN GRAPH”
STEP 3: choose a starting point as follows:
      3.1 If the number of odd vertices is 2, pick one of them (the first odd one)
      3.2 If there are no odd vertices pick any arbitrary vertex (the first one)
STEP 4: u = starting point
STEP 5: While there are edges in the graph, do:
      5.1 Find edges going out from u
            5.1.1 Always go for non-bridge edges
            5.1.2 If there are no non-bridge edges left, choose a bridge
      5.2 Store the other vertex of the edge (nextVertex)
      5.3 Remove the edge from graph
      5.4 nextVertex = u
STEP 6: Print the path
STEP 7: Stop
```

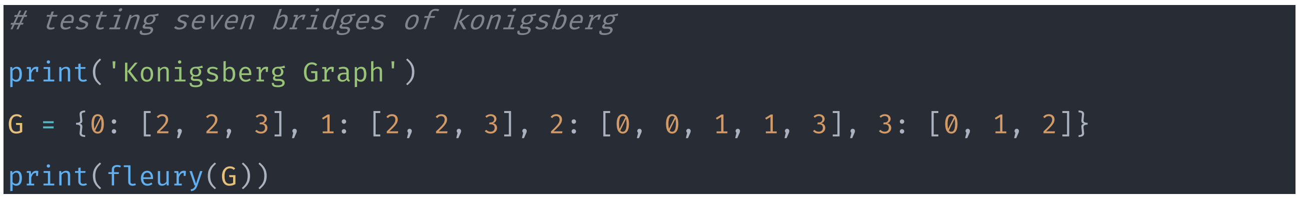
A.1.3 Code and results

Source Code

```
from pyvis.network import Network
from collections import defaultdict
from copy import copy

class Graph:
    # Constructor
    def __init__(self, edges, N):

        # A list of lists to represent an adjacency list
        self.adjList = defaultdict(lambda: [])

        # for visualizing the graph
        self.net = Network()
        self.net.add_nodes(list(range(N)))
        self.net.add_edges(edges)

        # add edges to the undirected graph
        for (src, dest) in edges:
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)

    def visualize(self, x):
        allEd = self.net.get_edges()
        for j in allEd:
            j.pop('color', None)
            j.pop('width', None)
        ctr = ord('a')-1
        for edge in x:
            for j in allEd:
                if (j['from'] == edge[0] and j['to'] == edge[1]) or (j['from'] == edge
[1] and j['to'] == edge[0]):
                    ctr += 1
                    j['color'] = 'rgba(0, 196, 154,0.5)'
                    j['width'] = 5
                    j['label'] = chr(ctr)


# is_connected - Checks if a graph in the form of a dictionary is
# connected or not, using Breadth-First Search Algorithm (BFS)
def is_connected(G):
    start_node = list(G)[0]
    color = {v: 'white' for v in G}
    color[start_node] = 'gray'
    S = [start_node]
    while len(S) != 0:
        u = S.pop()
        for v in G[u]:
            if color[v] == 'white':
                color[v] = 'gray'
                S.append(v)
            color[u] = 'black'
    return list(color.values()).count('black') == len(G)

# odd_degree_nodes - returns a list of all G odd degrees nodes
def odd_degree_nodes(G):
    odd_degree_nodes = []
    for u in G:
        if len(G[u]) % 2 != 0:
            odd_degree_nodes.append(u)
    return odd_degree_nodes


# from_dict - return a list of tuples links from a graph G in a dictionary format
def from_dict(G):
    links = []
    for u in G:
        for v in G[u]:
            links.append((u, v))
    return links


# fleury(G) - return eulerian trail from graph G or a
# string 'Not Eulerian Graph' if it's not possible to trail a path
def fleury(G):

    # checks if G has eulerian cycle or trail
    odn = odd_degree_nodes(G)
    if len(odn) > 2 or len(odn) == 1:
        return 'Not Eulerian Graph'
    else:
        g = copy(G)
        trail = []
        if len(odn) == 2:
            u = odn[0]
        else:
            u = list(g)[0]
        while len(from_dict(g)) > 0:
            current_vertex = u
            for u in g[current_vertex]:
                g[current_vertex].remove(u)
                g[u].remove(current_vertex)
                bridge = not is_connected(g)
                if bridge:
                    g[current_vertex].append(u)
                    g[u].append(current_vertex)
                else:
                    break
            if bridge:
                g[current_vertex].remove(u)
                g[u].remove(current_vertex)
                g.pop(current_vertex)
            trail.append((current_vertex, u))
    return trail
```

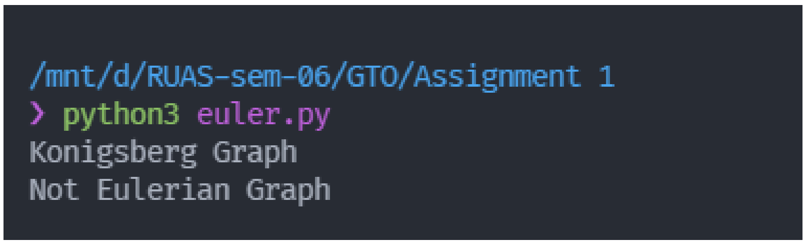
Results:

a. Testing with sample graph 1 (Konigsberg Graph):



```
# testing seven bridges of konigsberg
print('Konigsberg Graph')
G = {0: [2, 2, 3], 1: [2, 2, 3], 2: [0, 0, 1, 1, 3], 3: [0, 1, 2]}
print(fleury(G))
```

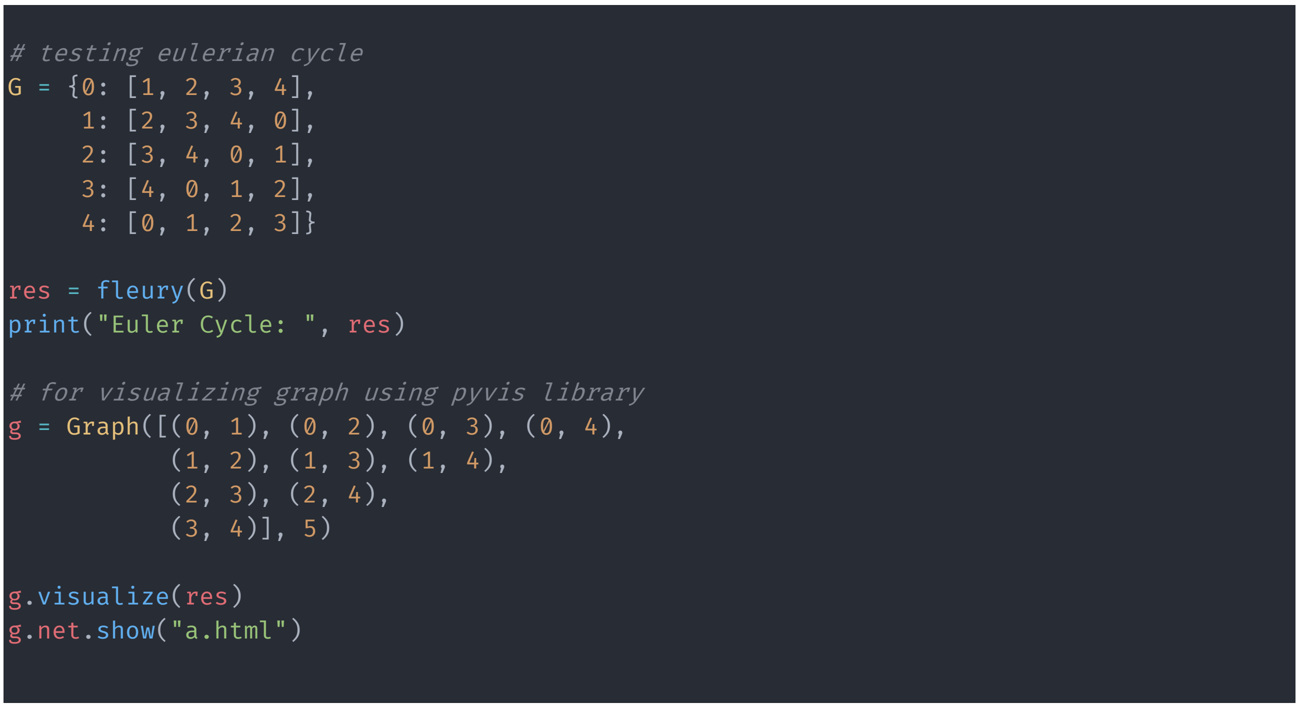
Figure 8 Input for sample 1 Konigsberg Graph



```
/mnt/d/RUAS-sem-06/GTO/Assignment 1
❯ python3 euler.py
Konigsberg Graph
Not Eulerian Graph
```

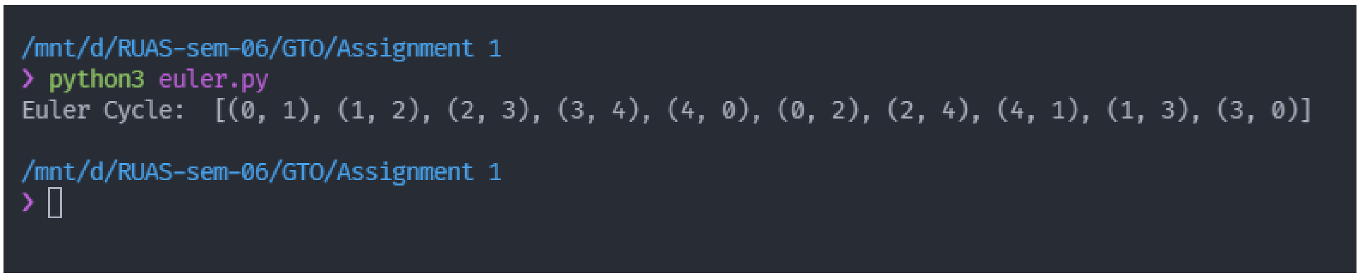
Figure 9 Output of Konigsberg Graph

b. Testing with sample graph 2



```
# testing eulerian cycle
G = {0: [1, 2, 3, 4],
     1: [2, 3, 4, 0],
     2: [3, 4, 0, 1],
     3: [4, 0, 1, 2],
     4: [0, 1, 2, 3]}

res = fleury(G)
print("Euler Cycle: ", res)

# for visualizing graph using pyvis library
g = Graph([(0, 1), (0, 2), (0, 3), (0, 4),
           (1, 2), (1, 3), (1, 4),
           (2, 3), (2, 4),
           (3, 4)], 5)

g.visualize(res)
g.net.show("a.html")
```

Figure 10 Testing with sample input 2



```
/mnt/d/RUAS-sem-06/GTO/Assignment 1
❯ python3 euler.py
Euler Cycle:  [(0, 1), (1, 2), (2, 3), (3, 4), (4, 0), (0, 2), (2, 4), (4, 1), (1, 3), (3, 0)]

/mnt/d/RUAS-sem-06/GTO/Assignment 1
❯ 
```

Figure 11 Output of sample

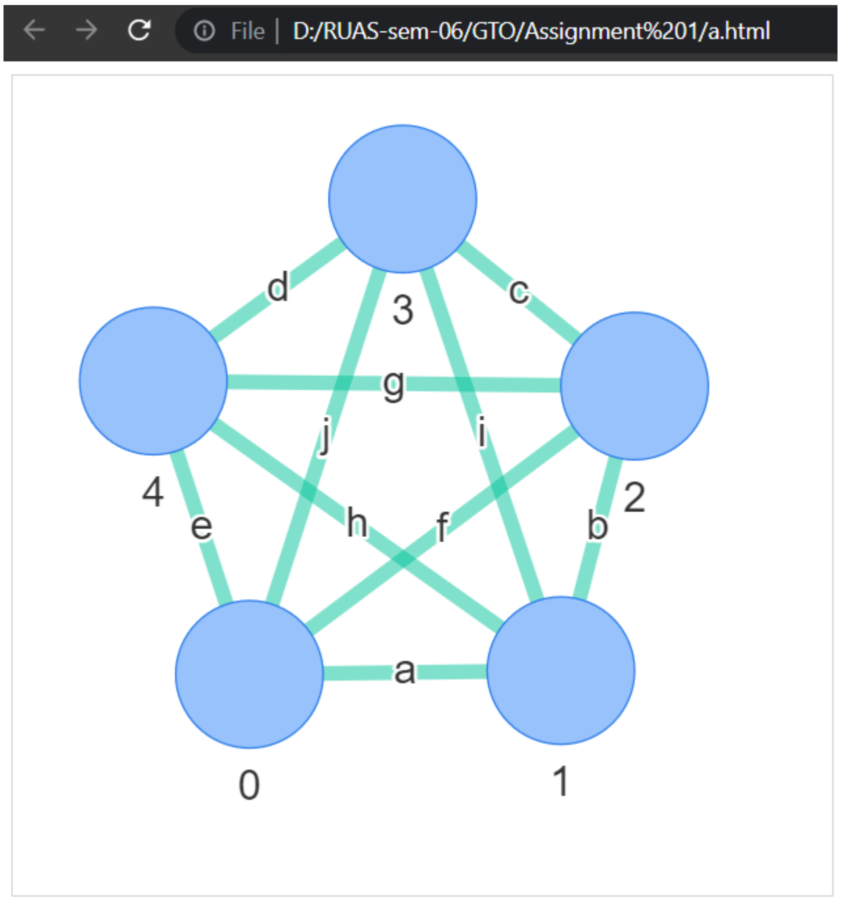


Figure 12 Visualization of Graph using pyvis library

Question No. 2

Solution to Question No. 2:

Sudoku is a single — player logic-based puzzle. A Sudoku puzzle is a grid of 81 cells, which is divided into 9 rows, columns and regions (or blocks).

The goal is to place the numbers from 1- 9 into empty cells in such a way, that in every row, every column and every region (3 x 3 block) each number appears only once.

Graph coloring is an assignment of different colors (or labels) to the vertices of a graph, such that no 2 adjacent (connected) vertices have the same color

In G- Graph Coloring Problem, we have to find if a graph can be colored with a minimum of 'G' colors. This 'G' is also known as the Chromatic Number of a Graph, and is denoted as χ(G)

B.1.1 Detailed Explanation of Approach

Sudoku Graph is a graph with 81 vertices (or nodes). Each cell in the Sudoku can be seen as a node of the graph.

Each node (or cell) has an edge to every other node (cell) in its respective column, row, and 3 x 3 grid.

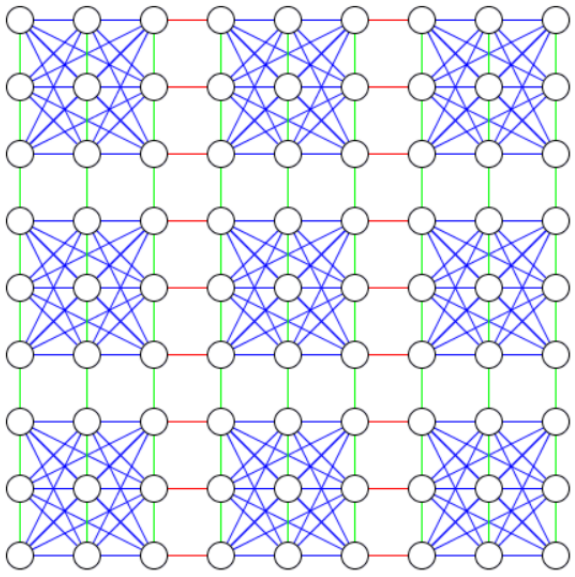


Figure 13 Visualizing Sudoku problem as a graph

Figure is a simplified image. As there are a lot of edges that are hidden. Each vertex has an edge from it to every vertex in its column and row, but in this image, all of these edges are lying on top of each other.

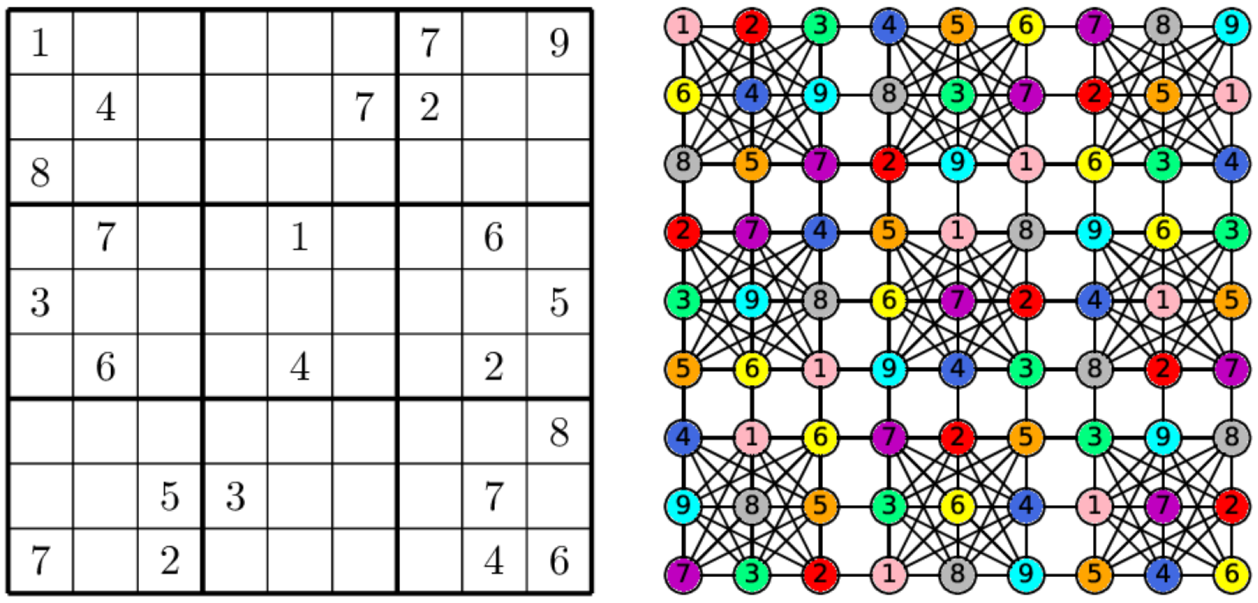
Please visit
https://www.reddit.com/r/dataisbeautiful/comments/6ty4vf/visualizing_the_sudoku_connectivity_graph_more_in/ or

https://preview.redd.it/tqfs8zpxlzfz.gif?format=mp4&s=4dd707b937cfc61f20864f89f7a6e7caa4e6c0f0

for visualizing more accurately.

So, we can say that Sudoku can be viewed as Graph and thus can be solved by Graph Coloring with a Chromatic Number, G = 9.

It is no different to using 9 different colors to color the vertices in a way that no two adjacent vertices have the same color.



1						7		9
	4				7	2		
8								
	7			1			6	
3								5
	6			4			2	
								8
		5	3				7	
7		2					4	6

(a) Unsolved Sudoku. (b) Graph coloring of Sudoku.

Figure 14 Graph coloring for 9x9 Sudoku puzzle

As it can be seen in Figure 14, all the numbers 1 through 9 are assigned a specific color, and no two vertices with the same color are connected, thus completing the objective of the puzzle.

For the sake of simplicity, let us consider the 4x4 Sudoku puzzle provided in Figure 15.

It comprises of 16 vertices and 56 edges in total.

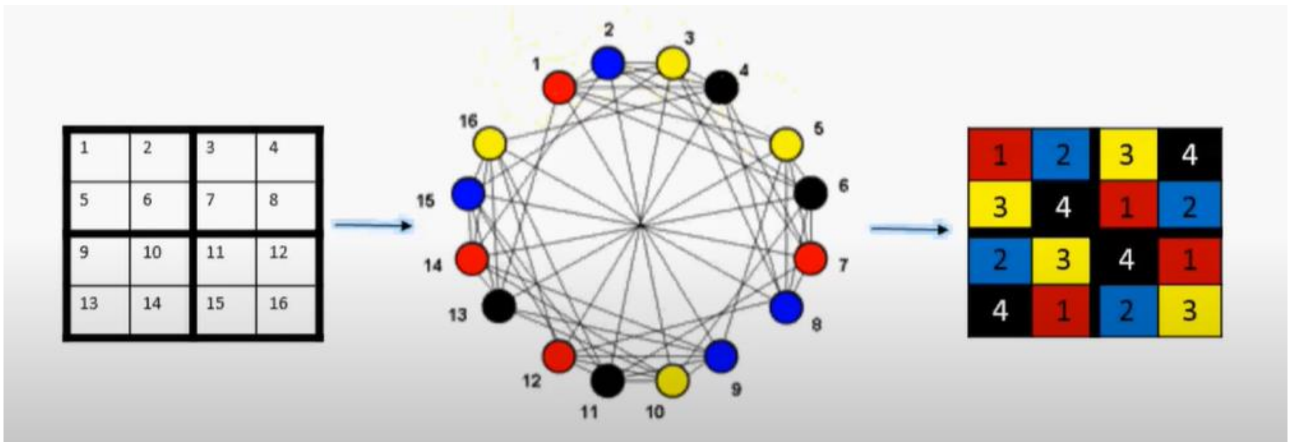


Figure 15 4x4 Sudoku puzzle solved using graph coloring

Considering only the first grid of the 4x4 Sudoku puzzle, it can be seen that 1 appears only once in the whole grid and is adjacent to the entire row containing the other 3 numbers/colors.

It can also be seen that it is also adjacent to the grid it is placed in.

B.1.2 Algorithm

There are 3 different basic problems in graph coloring:

- Graph is Given. Set of colors is given. Find out all ways in which that graph can be colored using the given colors.
- Graph is given. Set of colors is given. Find out if the graph can be colored using the given set of colors. (m -Coloring Decision Problem)
- Graph is given. Set of colors is not given. Find the minimum number of colors required to color the given graph (m — Coloring Optimization Problem)

Algorithm for Graph Coloring (m-Coloring Decision Problem)

The idea is to assign colors one by one to different vertices, starting from the vertex 0. Before assigning a color, check for safety by considering already assigned colors to the adjacent vertices i.e. check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false.

```
STEP 1: Start
STEP 2: Create a recursive function that takes current vertex index,
number of vertices and output color array as arguments.
STEP 3: If the current vertex index is equal to number of vertices.
Return True and print the color configuration in output array.
STEP 4: Assign color to a vertex (1 to m).
STEP 5: For every assigned color, check if the configuration is safe,
(i.e., check if the adjacent vertices do not have the same color)
recursively call the function with next index and number of vertices
STEP 6: If any recursive function returns true break the loop and
return true.
STEP 7: If no recursive function returns true, then return false.
STEP 8: Stop
```

Using greedy coloring algorithm

```
STEP 1: Start
STEP 2: Color a vertex with color 1
STEP 3: Pick an uncolored vertex v.
     3.1 Color it with the lowest-numbered color that has not been
used on any previously-colored vertices adjacent to v.
     3.2 If all previously-used colors appear on vertices adjacent to
v, this means that we must introduce a new color and number it.
STEP 4: Repeat the Step 3 until all vertices are colored.
STEP 5: Stop
```

The implementation of this Sudoku solver using Python3 can be viewed at

https://nbviewer.jupyter.org/github/subhendu17620/RUAS-sem-06/blob/main/GTO/Assignment%201/Sudoku.ipynb

1. https://medium.com/code-science/sudoku-solver-graph-coloring-8f1b4df47072
2. https://www.reddit.com/r/dataisbeautiful/comments/6ty4vf/visualizing_the_sudoku_connectivity_graph_more_in/
3. http://pi.math.cornell.edu/~mec/Summer2009/meerkamp/Site/Sudokus_as_Graphs.html
4. http://www.cs.kent.edu/~dragan/ST-Spring2016/SudokuGC.pdf