RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT - 1

Course Code	19CSC311A
Course Name	Graph Theory and Optimization
Programme	B.Tech
Department	CSE
Faculty	FET

Name of the Student	K Srikanth
Reg. No	17ETCS002124
Semester/Year	6th / 3rd Year
Course Leader/s	Mr. Narasimha Murthy K. R.

Declaration Sheet				
Student Name	K Srikanth			
Reg. No	17ETCS002124			
Programme	B.Tech		Semester/Year	6th/3rd Year
Course Code	19CSC311A			
Course Title	Graph Theory and Optimization			
Course Date	18/03/2021	to	06/07/2021	
Course Leader	Mr. Narasimha Murthy K. R.			

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	K Srikanth	Date	02-06-2021
Submission date stamp (by Examination & Assessment Section)			

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

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech. in CSE
Semester/Batch	06/2018		
Course Code	19CSC311A	Course Title	Graph Theory and Optimization
Course Leader	Ms. Pallavi R. Kumar and Mr. Narasimha Murthy K. R.		

Assignment-1			
Reg. No.	17ETCS002124	**Name of Student**	K Srikanth

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**

Please note:

1. Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.
2. The First Examiner is required to mark the comments in RED ink and the Second Examiner's comments should be in GREEN ink.
3. The marks for all the questions of the assignment have to be written only in the **Component – CET B: Assignment** table.
4. If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.

Assignment 1

Term - 1

Instructions to students:

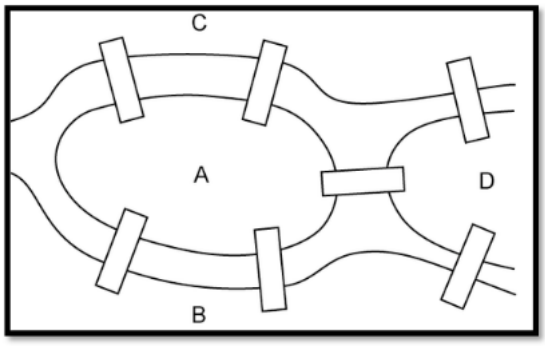
1. The assignment consists of **2** questions: Part A-**1** Question, Part B-**1** Question.
2. Maximum marks is **25**.
3. The assignment has to be neatly word processed as per the prescribed format.
4. The maximum number of pages should be restricted to **10**.
5. The printed assignment must be submitted to the course leader.
6. **Submission Date: 2nd June 2021**
7. **Submission after the due date is not permitted.**
8. **IMPORTANT**: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.
9. Marks will be awarded only to the sections and subsections clearly indicated as per the problem statement/exercise/question

Name: K Srikanth Registration Number: 17ETCS002124

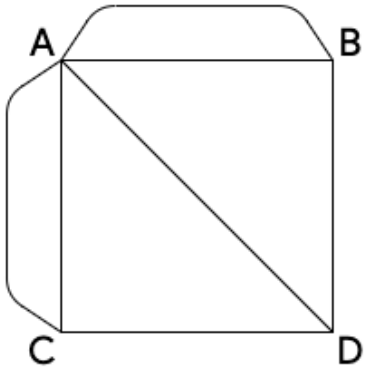
Part A

Q-A1.1

Given,



Now let's convert the given image into a graph, with all the lands as vertices and bridges as edges



Note

- A graph has an Euler circuit if and only if the degree of every vertex is even.
- A graph has an Euler path if and only if there are at most two vertices with odd degree.

Name: K Srikanth Registration Number: 17ETCS002124

Q-A1.1-a

Can the people of Konigsberg successfully walk over all the bridges once and get?

back to where they started?

Answer: No, they can't travel through all the bridges once and get back to where they started as the graph, **we considering is as a Euler Graph**

Justification:

The Degree of the Vertices i.e. **A = 5, B = 3, C = 3 and D = 3** but to find the **circuit to return back to same point** the Degree of the Vertices should be even not odd in order to find euler circuit

Q-A1.1-b

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

Answer: No, the bridge walk cannot be achieved if the people were happy not returning to their starting point as the graph, **we considering is as a Euler Graph**

Justification:

The Degree of the Vertices i.e., **should be odd and total number of odd vertices should be <= 2.** Clearly, all the vertices are odd so there are **4 odd vertices in the given graph no path is possible.**

Q-A1.1-c

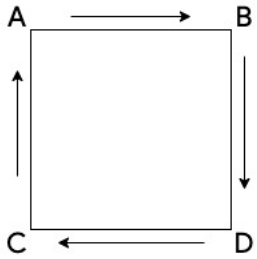
If one or more of the bridges were removed, can the round trip walk around the bridges of Konigsberg be achieved?

Answer: Yes, the round bridge walk can be achieved if we remove 3 edges from the existing graph and get back to where they started as the graph, **we considering is as a Euler Graph**

Justification:

Given Graph, The Degree of the Vertices i.e. **A = 5, B = 3, C = 3 and D = 3** but to find the **circuit to return back to same point** the Degree of the Vertices should be even not odd in order to find euler circuit, (Figure 2)

Updated Graph



As you can that we have removed the parallel edges between a vertices AB and AC and removed a diagonal edge from AD now the Graph as a Euler Circuit as, The Degree of the Vertices i.e. **A = 2, B = 2, C = 2 and D = 2**

Q-A1.1-d

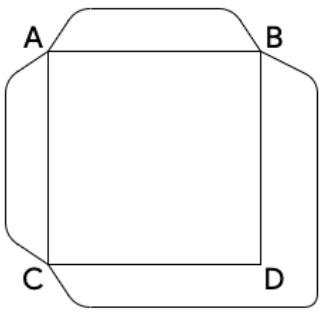
If the city of Konigsberg had seven bridges arranged in some other way, will it be possible to make the round trip walk successfully?

Answer: Yes, the round bridge walk can be achieved if we modify the edge that is traversing from AD from the existing graph to BC in the Updated Graph and get back to where they started as the graph, **we considering is as a Euler Graph**

Justification:

The Degree of the Vertices i.e. **A = 5, B = 3, C = 3 and D = 3** but to find the **circuit to return back to same point** the Degree of the Vertices should be even not odd in order to find euler circuit (Figure 2)

Updated Graph



As you can see, we have **updated the graph and replaced the vertex from AD from the original graph to BC** in the new graph. Now, in order to **find the circuit**, we have to look at the **Degree of the Vertices** and they have to be even so, **A = 4, B = 4, C = 4 and D = 2**

Now the updated graph has a Euler Circuit.

Name: K Srikanth Registration Number: 17ETCS002124

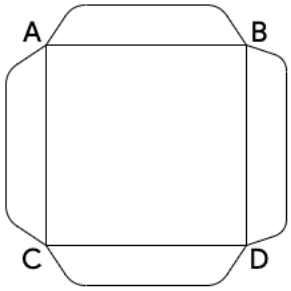
Q-A1.1-e

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

Answer: Yes, the round bridge walk can be achieved if we modify the edge that is traversing from AD from the existing graph to BC in the Updated Graph and get back to where they started as the graph, **we considering is as a Euler Graph**

Justification:

Case 1: If the seven bridges of Konigsberg **are replaced with Eight, yes, the Euler Circuit and Euler Path are possible.**



Condition:

Euler Circuit:

All the vertices degree should be even, right? So, in the new graph **A = 4, B = 4, C = 4 and D = 4.** So, this has a **Euler Circuit**

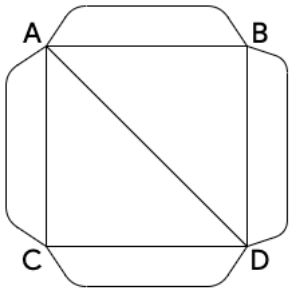
Euler Path:

The Sum of odd degree should be less than or equal 2**.. Here in there are 0.** So this has a **Euler Path**

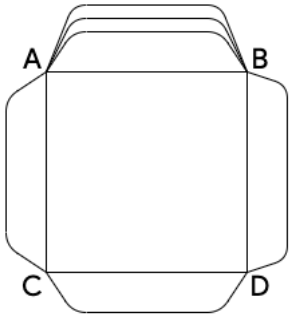
Case 2: If the seven bridges of Konigsberg **are replaced with Nine. The Euler Circuit is not possible but Euler Path is possible.**

Euler Path:

The Sum of odd degree should be less than or equal 2**. Here in there are 2.** So, this has a **Euler Path as Vertices A and D are odd vertices with degree 5.**



Case 3: If the seven bridges of Konigsberg **are replaced with Ten, yes, the Euler Circuit and Euler Path are possible.**



Condition:

Euler Circuit:

All the vertices degree should be even, right? So, in the new graph **A = 6, B = 6, C = 4 and D = 4.** So, this has a **Euler Circuit**

Euler Path:

The Sum of odd degree should be **less than or equal 2. Here in there are 0.** So, this has a **Euler Path**

Name: K Srikanth Registration Number: 17ETCS002124

Q-A1.1-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?

Answer: To generalized result of n number of bridges we can. So the condition is **If we have degrees of odd number of vertices and even number of vertices the we can generalise it,**

$$\sum_{even} d(v_i) + \sum_{odd} d(v_j) = \sum_{i=1}^{n} d(v_i)$$

Where **"d(vi)"** is **even number of edge at a vertex** and **"d(vj)"** is **odd number of edge at a vertex.** Now in order to find the number of bridges in a general form it would be,

$$\sum_{i=1}^{n} d(v_i) = 2\ E$$

where "E" is No of Bridges.

Given the Initial Graph we can find out how many vertices are there and how many edges by constructing an adjacency matrix,

	A	B	C	D
A	0	2	2	1
B	2	0	0	1
C	2	0	0	1
D	1	1	1	0

Name: K Srikanth Registration Number: 17ETCS002124

Q-A1.2

Q-A1.2-a

Algorithm

1. Start
2. Declare a Class (Graph)
    a. Declaring Instance of the class vertices (v)
    b. Declaring Instance of the class graph(* which is a defaultdict)
    c. Declaring Instance of the class time
    d. Declaring Instance of the class last_reached.
3. Declaring Function **(addBridge)** with params (u,v)
    a. Add(v) to graph (u)
4. Declaring Function **(removeBridge)** with params (u,v)
    a. For loop begins (index, key in enumerate of graph(element))
        i. If condition (key == v)
            1. Pop the element from the graph
    b. For loop begins (index, key in enumerate of graph(element))
        i. If condition (key == u)
            1. Pop the element from the graph
5. Declaring Function **(DFC Count)** with params (v, visited)
    a. Initialize count equals to 1
    b. Make visited of element true
    c. For loop beings (for element in graph visited)
        i. If visited element then make it false
            1. count += Instance of DFSCount(i, visited)
    d. Return count
6. Declaring Function (**isNextEdgeValid)** with params (u,v)
    a. If length of graph[u] is 1
        i. Return true
    b. Else
        i. Make Visisted false for all the instances of V

ii. Count1 += Instance of DFSCount(i, visited)
iii. Remove the bridge at (u,v)
iv. Make Visisted false for all the instances of V
v. Count2 += Instance of DFSCount(i, visited)
vi. Add the bridge at (u,v)
vii. Return false if count 1 > count 2 else make it true

7. Declaring function (travel) with prams (u)
    a. Initialize instance of last_reached as u
    b. For Loop Begins (v in graph [u])
        i. If is isNextEdgeValid (u,v)
            1. Display the path
            2. Remove the Bridge
            3. Continue traveling
8. Declaring function (printpath)
    a. Initializing u as 0
    b. For loop begins (in in v)
        i. Check If length of (graph [element] % 2 !=0)
            1. If yes then make u = element then break
    c. Calling instance of travel (u)
    d. If not u == last reached
        i. Display (we didn't make it back to the point)
    e. Else
        i. Display (We made it back to the point)
9. Declare the graph object with its instances to create a bridge with addBridge function
10. Stop

Q-A1.2-b

Code

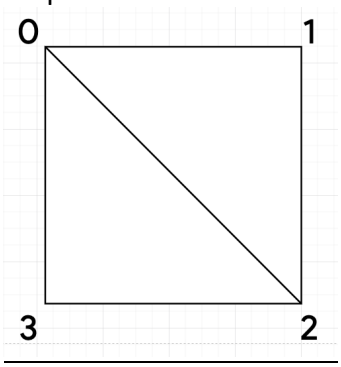
```python
# 17ETCS002124 K Srikanth
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = defaultdict(list)
        self.Time = 0
        self.last_reached = 0

    def addBridge(self, u, v):
        self.graph[u].append(v)

    def removeBridge(self, u, v):
        for index, key in enumerate(self.graph[u]):
            if key == v:
                self.graph[u].pop(index)
        for index, key in enumerate(self.graph[v]):
            if key == u:
                self.graph[v].pop(index)

    def DFSCount(self, v, visited):
        count = 1
        visited[v] = True
        for i in self.graph[v]:
            if visited[i] == False:
                count = count + self.DFSCount(i, visited)
        return count

    def isNextEdgeValid(self, u, v):
        if len(self.graph[u]) == 1:
            return True
        else:
            visited = [False] * (self.V)
            count1 = self.DFSCount(u, visited)
            self.removeBridge(u, v)
            visited = [False] * (self.V)
            count2 = self.DFSCount(u, visited)
            self.addBridge(u, v)
            return False if count1 > count2 else True

    def travel(self, u):
        self.last_reached = u
        for v in self.graph[u]:
            if self.isNextEdgeValid(u, v):

                print("%d-%d " % (u, v)),
                self.removeBridge(u, v)
                self.travel(v)

    def printPath(self):
        u = 0
        for i in range(self.V):
            if len(self.graph[i]) % 2 != 0:
                u = i
                break
        print("\n")
        self.travel(u)
        if not u == self.last_reached:
            print(
                "we made the tour across all the landmasses \nwe could not comeback to where we started off :("
            )
        else:
            print(
                "we made the tour across all the landmasses \nwe have come back to where we started off :)"
            )


print("\n")
print("******** 17ETS002124 K Srikanth ********")
g1 = Graph(4)
g1.addBridge(0, 1)
g1.addBridge(1, 2)
g1.addBridge(2, 3)
g1.addBridge(3, 0)
g1.addBridge(0, 2)
g1.printPath()

g2 = Graph(4)
g2.addBridge(0, 1)
g2.addBridge(1, 2)
g2.addBridge(2, 3)
g2.addBridge(3, 0)
g2.printPath()
```

Python Code for the given problem statement

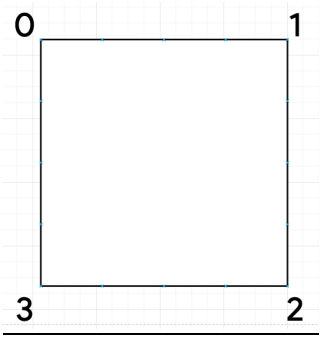
Name: K Srikanth Registration Number: 17ETCS002124

Output

Here the Graph for G1 is, as an example is



Here the Graph for G2 is, as an example is



Code Output

```
******** 17ETS002124 K Srikanth ********


1-2
2-3
3-0
we made the tour across all the landmasses
we could not comeback to where we started off :(


0-1
1-2
2-3
3-0
we made the tour across all the landmasses
we have come back to where we started off :)

~ >
```

Python Code output for the given problem statement

Name: K Srikanth Registration Number: 17ETCS002124

Part B

Q-B1.1

To Solve Sudoku using graph theory using coloring, we have to setup some ground rules in order to achieve that,

Rule 1: Let's make sure that all the numbers in the sudoku puzzle i.e. {1,2,3,4,5,6,7,8,9} have independent color. So,

Let's assume,

1. Number 1 is **Yellow**
2. Number 2 is **Green**
3. Number 3 is **Blue**
4. Number 4 is **Purple**
5. Number 5 is **Red**
6. Number 6 is **Black**
7. Number 7 is **Orange**
8. Number 8 is **Cyan**
9. Number 9 is **Magenta**

Rule 2: Now that we have organized our colors for our numbers. The Rule in order to solve our Sudoku is to make sure that that each number (i.e. color) from above should not be mapped to the same element in a row or column or the 3 X 3 Box that its currently in the state in

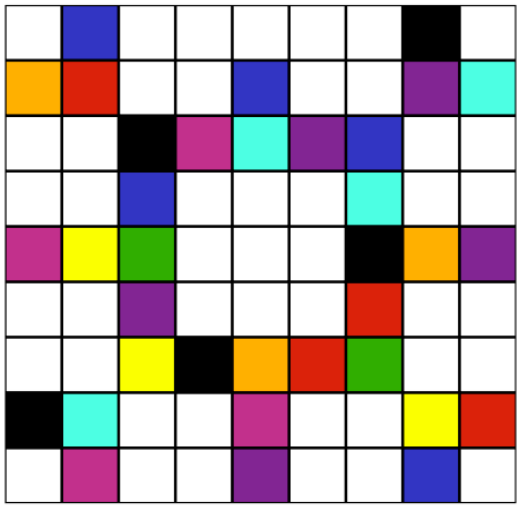
I.e.

Let's take an example of a sudoku problem

Given,

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

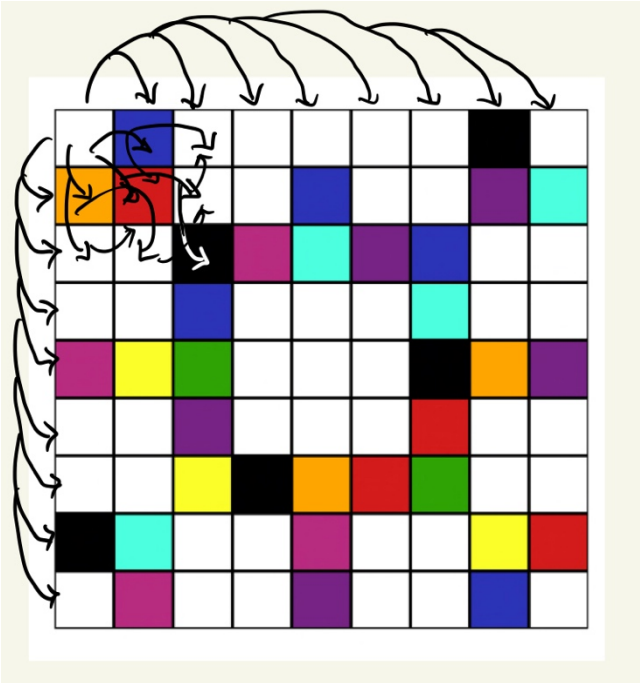
After Coloring it with Numbers,



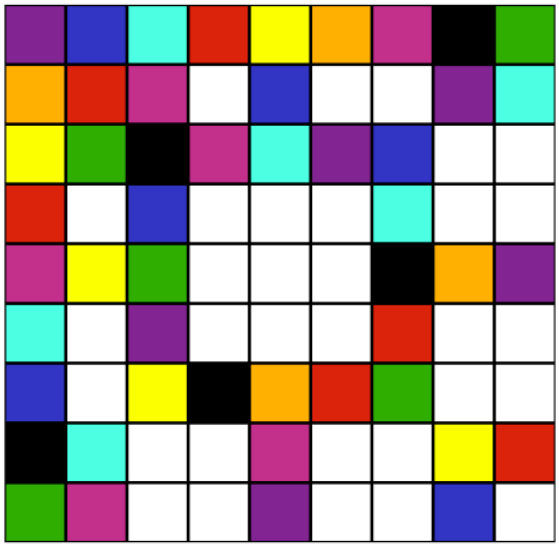
After we are done coloring out our number from the given problem statement the next thing, we do is map (make an edge) it. With its row and column and its 3 X 3 Box. We have to ensure that we don't want to connect to the same color again as our connected nodes should be different from one and other.

Name: K Srikanth Registration Number: 17ETCS002124

Solving the First box,



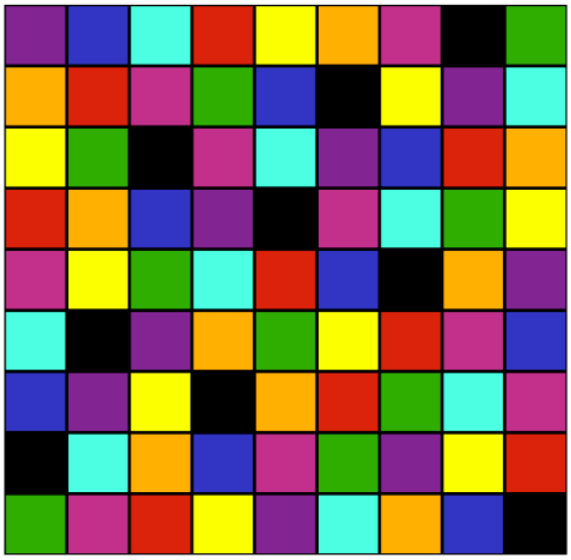
After connecting all the edges from one vertex our coloring graph will look something similar to this no, we just have to color the rest of the boxes in a way that don't match up with the parent node that it is connected to with different colors. After coloring the First Box the Puzzle looks something like this,



Now that we have colored our first box where each color from the node is different when it is connected to other node.

And so, on we repeat our process over and over again to color everything out using this technique

End Result



Hence, we solved our sudoku problem with graph coloring technique

Q-B1.2,

Algorithm

1. **Start**
2. Create a recursive function that takes current vertex index, number of vertices and output colour array as arguments.
3. If the current vertex index is equal to number of vertices. Return True and print the colour configuration in output array.
4. Assign colour to a vertex (1 to 9).
5. For every assigned colour, check if the configuration is safe, (**i.e. check if the adjacent vertices do not have the same colour)** recursively call the function with next index and number of vertices
6. If any recursive function returns true break the loop and return true.
7. If no recursive function returns true, then return false.
8. **Stop**