# TASK:6

Solve a **Map Coloring problem** using constraint satisfaction approach by applying following constraints

**Aim:** To Solve a Map Coloring problem using constraint satisfaction approach using Graph online and visualago online simulator

**Algorithm:**

**Step 1:** Confirm whether it is valid to color the current vertex worth the current color (by checking whether any of its adjacent vertices are colored with the same color)

**Step 2:** If yes then color it and otherwise try a different color

**Step 3:** check if all vertices are colored or not

**Step 4:** If not then move to the next adjacent uncolored vertex

**Step 5:** Here backtracking means to stop further recursive calls on adjacent vertices.

**Program:**

class Graph:

def \_\_init\_\_(self, vertices):

self.v = vertices

self.graph = [[0 for \_ in range(vertices)] for \_ in range(vertices)]

def is\_safe(self, v, color, c):

for i in range(self.v):

if self.graph[v][i] == 1 and color[i] == c:

return False

return True

def graph\_color\_util(self, m, color, v):

if v == self.v:

return True

for c in range(1, m + 1):

if self.is\_safe(v, color, c):

color[v] = c

if self.graph\_color\_util(m, color, v + 1):

return True

color[v] = 0

return False

def graph\_coloring(self, m):

color = [0] \* self.v

if not self.graph\_color\_util(m, color, 0):

return False

print("Solution exists and following are the assigned colors:")

for c in color:

print(c, end=" ")

return True

if \_\_name\_\_ == '\_\_main\_\_':

g = Graph(4)

g.graph = [

[0, 1, 1, 1],

[1, 0, 1, 0],

[1, 1, 0, 1],

[1, 0, 1, 0]

]

m = 3

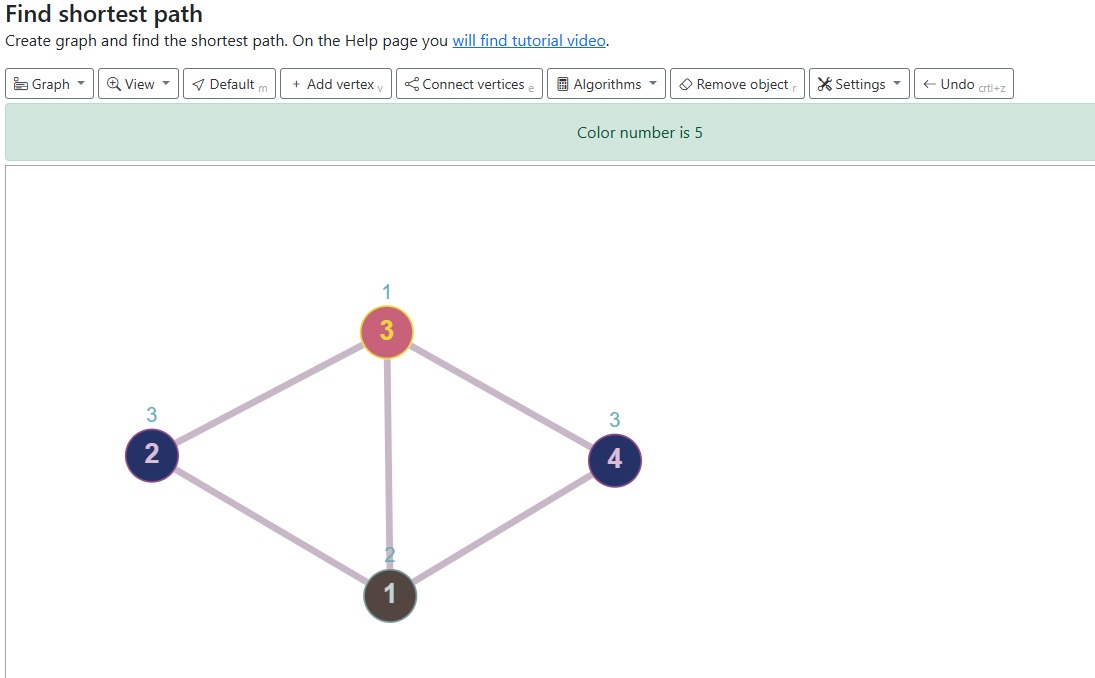
g.graph\_coloring(m)

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**GRAPH:**



**Result:**

Thus Solving a Map Coloring problem using constraint satisfaction approach using Graphonline and visulago online simulator was successfully executed and output was verified.