PROGRAM 11:

#Number of vertices in the graph

# define 4 4

# check if the colored

# graph is safe or not

def isSafe(graph, color):

# check for every edge

for i in range(4):

for j in range(i + 1, 4):

if (graph[i][j] and color[j] == color[i]):

return False

return True

# /\* This function solves the m Coloring

# problem using recursion. It returns

# false if the m colours cannot be assigned,

# otherwise, return true and prints

# assignments of colours to all vertices.

# Please note that there may be more than

# one solutions, this function prints one

# of the feasible solutions.\*/

def graphColoring(graph, m, i, color):

# if current index reached end

if (i == 4):

# if coloring is safe

if (isSafe(graph, color)):

# Print the solution

printSolution(color)

return True

return False

# Assign each color from 1 to m

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

color[i] = j

# Recur of the rest vertices

if (graphColoring(graph, m, i + 1, color)):

return True

color[i] = 0

return False

# /\* A utility function to print solution \*/

def printSolution(color):

print("Solution Exists:" " Following are the assigned colors ")

for i in range(4):

print(color[i],end=" ")

# Driver code

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

# /\* Create following graph and

# test whether it is 3 colorable

# (3)---(2)

# | / |

# | / |

# | / |

# (0)---(1)

# \*/

graph = [

[ 0, 1, 1, 1 ],

[ 1, 0, 1, 0 ],

[ 1, 1, 0, 1 ],

[ 1, 0, 1, 0 ],

]

m = 3 # Number of colors

# Initialize all color values as 0.

# This initialization is needed

# correct functioning of isSafe()

color = [0 for i in range(4)]

if (not graphColoring(graph, m, 0, color)):

print ("Solution does not exist")

OUTPUT:

