Graph Coloring Problem’s Solution Using Backtracking Algorithm:

V = 4

# A function to print the color configuration.

def printConfiguration(colorArray):

print("The assigned colors are as follows:")

for i in range(4):

print("Vertex: ",

i, " Color: ", colorArray[i])

"""

A function that will check if the current colorArray of the graph is safe or not.

"""

def isSafe(v, colorArray, vertex):

for i in range(V):

if graph[v][i] == 1 and colorArray[i] == vertex:

return False

return True

"""

A recursive function that takes the current index, number of vertices, and the color array. If the recursive call returns true then the coloring is possible. It returns

false if the m colors cannot be assigned.

"""

def graphColoringAlgorithmUtil(m, colorArray, currentVertex):

# base case.

if currentVertex == V:

return True

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

if isSafe(currentVertex, colorArray, i) == True:

colorArray[currentVertex] = i

if graphColoringAlgorithmUtil(m, colorArray, currentVertex + 1):

return True

# backtrack

colorArray[currentVertex] = 0

def graphColoringAlgorithm(colorArray, m):

# Initially the color array is initialized with 0.

colorArray = [0] \* V

# Call graphColoringAlgorithmUtil() for vertex 0.

if graphColoringAlgorithmUtil(m, colorArray, 0) == None:

print("Coloring is not possible!")

return False

# Print the solution

print("Coloring is possible!")

printConfiguration(colorArray)

return True

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

graph = [

[0, 1, 1, 1],

[1, 0, 1, 0],

[1, 1, 0, 1],

[1, 0, 1, 0],

]

m = 3

graphColoringAlgorithm(graph, m)

**#Output:**

**Coloring is possible!**

**The assigned colors are as follows:**

**Vertex: 0 Color: 1**

**Vertex: 1 Color: 2**

**Vertex: 2 Color: 2**

**Vertex: 3 Color: 2**