Input Format:

First line contains n, number of nodes.

Next line contains m, number of colors.

Next line contains e, number of edges.

Next e lines contain u, v which denotes an undirected edge between u and v.

**Algorithm 1 (Naïve Recursive)**

* We have an adjacency matrix named “graph” of size n x n.
* We also have an array named “color” of size n of size n initialized to 0.
* We have a function printSolution to print the array “color”

graphColoring(int graph[V][V], int m, int i, int color[V] )

1. if i = V then
   1. if isSafe(graph,color) = true
      1. printSolution(color)
      2. return true
   2. return false
2. for j= 1 to m
   1. set color[i] = j
   2. if graphColoring(graph, m, i+1, color) = true
      1. return true
   3. set color[i] = 0
      1. return true
3. return false

isSafe(int graph[V][V], int color[])

1. for i = 0 to V-1
   1. for j = i +1 to V-1
      1. if graph[i][j] = true and color[j] = color[i]
         1. return false
2. return true

Driver Function

1 call graphColoring(graph, m, 0, color)

2 if it returns false then print(“Solution does not exit”)

**Algorithm 2 (Backtracking)**

* We have an adjacency matrix named “graph” of size n x n.
* We also have an array named “color” of size n initialized to 0.
* We have a function printSolution to print the array “color”

colorGraphRecursive(int graph[V][V], int m, int i, int color[V] )

1. if i = V then
   1. return true
2. for c = 1 to m
   1. if isSafe(graph, color, i, c)
      1. color[i] = c
      2. if colorGraphRecursive (graph, m, i + 1, color) = true
         1. return true
      3. color[i] = 0
3. return false

isSafe(int graph[V][V], int color[],int v, int c)

1. for i = 0 to V-1
   1. if graph[v][i] = true and c = color[i]
      1. return false
2. return true

Driver Function

1 if colorGraphRecursive (graph, m, 0, color) = false

* 1. print(“Solution does not exist”)

2 else

* 1. call printSolution(color)

**Algorithm 2 (BFS Approach)**

* We have a class named “node” which has properties “color” which is initially equal to 1 and “edges” which is adjacency list.
* We also have a vector named “visited” of size n initialized to 0.
* We have a function printSolution to print the array “color”

graphColoring(vector<node>& nodes, int n, int m)

1. set maxColors = 1

2. for i = 1 to n

a. if visited[i] = 1

i. continue

b. visited[i] = 1

c. declare queue named “q”

d. push i in q

e. while q is not empty

i. set top = q.front()

ii. pop element from queue

iii. for e in node[top].edges

* + - 1. if nodes[top].color = nodes[e].color = true
         1. nodes[e].color += 1
      2. maxColors = max(maxColors,nodes[top].color,nodes[e].color)
      3. if maxColors > m
         1. return 0
      4. if visited[e] = false
         1. visited[e] = 1
         2. push e in q

3. return 0