119. Graph Coloring

AIM: To solve the Graph Coloring problem by using backtracking

PROGRAM:

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adj_matrix = [[0]*vertices for _ in range(vertices)]
        self.colors = [0] * vertices  # Colors assigned to vertices, initialized to 0 (unassigned)

    def is_safe(self, v, c):
        """ Check if it's safe to color vertex v with color c """
        for i in range(self.vertices):
            if self.adj_matrix[v][i] == 1 and self.colors[i] == c:
                return False
        return True

    def graph_coloring_util(self, m, v):
        """ Recursive utility function to solve graph coloring """
        if v == self.vertices:
            return True

        for c in range(1, m+1):
            if self.is_safe(v, c):
                self.colors[v] = c
                if self.graph_coloring_util(m, v + 1):
                    return True
                self.colors[v] = 0  # Backtrack
```

```
    def graph_coloring(self, m):
        """ Function to solve graph coloring problem """
        if not self.graph_coloring_util(m, 0):
            print("No solution exists with {} colors.".format(m))
            return False

        print("Solution found with {} colors:".format(m))
        print("Vertex   Color")
        for i in range(self.vertices):
            print(f"{i + 1}        {self.colors[i]}")

# Example usage:
g = Graph(4)
g.adj_matrix = [
    [0, 1, 1, 1],
    [1, 0, 1, 0],
    [1, 1, 0, 1],
    [1, 0, 1, 0]
]
m = 3
g.graph_coloring(m)
```

OUTPUT:

```
Solution found with 3 colors:
Vertex   Color
1        1
2        2
3        3
4        2
```

TIME COMPLEXITY: $O(m^v)$