**M – Coloring Problem**

**Problem Statement:** Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.
**Examples:**

**Example 1:**

**Input:**

N = 4

M = 3

E = 5

Edges[] = {

(0, 1),

(1, 2),

(2, 3),

(3, 0),

(0, 2)

}

**Output:** 1

**Explanation:** It is possible to colour the given graph using 3 colours.

**Example 2:**

**Input:**

N = 3

M = 2

E = 3

Edges[] = {

(0, 1),

(1, 2),

(0, 2)

}

**Output:** 0

**Explanation:** It is not possible to color.

**Solution**

***Disclaimer***: *Don't jump directly to the solution, try it out yourself first.*
**Solution 1: Backtracking**
**Approach**:
Basically starting from vertex 0 color one by one the different vertices.
**Base condition:**
If I have colored all the N nodes return true.
**Recursion:**
Trying every color from 1 to m with the help of a for a loop.
Is safe function returns true if it is possible to color it with that color i.e none of the adjacent nodes have the same color.

In case this is an algorithm and follows a certain intuition, please mention it.
Color it with color i then call the recursive function for the next node if it returns true we will return true.
And If it is false then take off the color.
Now if we have tried out every color from 1 to m and it was not possible to color it with any of the m colors then return false.
**Code:**

- C++ Code
- Java Code
- Python Code

```cpp
#include<bits/stdc++.h>

using namespace std;
bool isSafe(int node, int color[], bool graph[101][101], int n, int col) {
   for (int k = 0; k < n; k++) {
      if (k != node && graph[k][node] == 1 && color[k] == col) {
         return false;
      }
```

```
    }
    return true;
}
bool solve(int node, int color[], int m, int N, bool graph[101][101]) {
    if (node == N) {
        return true;
    }

    for (int i = 1; i <= m; i++) {
        if (isSafe(node, color, graph, N, i)) {
            color[node] = i;
            if (solve(node + 1, color, m, N, graph)) return true;
            color[node] = 0;
        }

    }
    return false;
}

//Function to determine if graph can be coloured with at most M colours such
//that no two adjacent vertices of graph are coloured with same colour.
bool graphColoring(bool graph[101][101], int m, int N) {
    int color[N] = {
        0
    };
    if (solve(0, color, m, N, graph)) return true;
    return false;
}

int main() {
    int N = 4;
    int m = 3;

    bool graph[101][101];
    memset(graph, false, sizeof graph);

    // Edges are (0, 1), (1, 2), (2, 3), (3, 0), (0, 2)
    graph[0][1] = 1; graph[1][0] = 1;
    graph[1][2] = 1; graph[2][1] = 1;
    graph[2][3] = 1; graph[3][2] = 1;
    graph[3][0] = 1; graph[0][3] = 1;
    graph[0][2] = 1; graph[2][0] = 1;

    cout << graphColoring(graph, m, N);

}
```

**Output:** 1

**Time Complexity: O( N^M) (n raised to m)**

**Space Complexity: O(N)**