M-Coloring Problem - GeeksforGeeks

Source: https://www.geeksforgeeks.org/m-coloring-problem/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund M-Coloring Problem Last Updated : 23 Jul, 2025 Given an edges of graph and a number m , the your task is to check it is possible to color the given graph with at most m colors such that no two adjacent vertices of the graph are colored with the same color . Examples Input: V = 4, edges[][] = [[0, 1], [0, 2], [0,3], [1, 3], [2, 3]], m = 3 Output: true Explanation: Structure allows enough separation between connected vertices, so using 3 colors is sufficient to ensure no two adjacent vertices share the same color—hence, the answer is true Input: V = 5, edges[][] = [[0, 1], [0, 2], [0, 3], [1, 2], [1, 4], [2, 3], [2, 4], [3, 4]], m = 3 Output: false Explanation: In this graph, the vertices are highly interconnected, especially vertex 2, which connects to four others. With only 3 colors, it's impossible to assign colors so that no two adjacent vertices share the same color, hence, the answer is false . Try it on GfG Practice Approach 1: Generate all possible configurations - O((V + E)*m^V) Time and O(E+V) Space Generate all possible configurations of length V of colors. Since each node can be colored using any of the m available colors, the total number of color configurations possible is m V . After generating a configuration of color, check if the adjacent vertices have the same color or not . If the conditions are met, print the combination. C++ #include <bits/stdc++.h> using namespace std ; bool goodcolor (vector < int > adj [], vector < int > col){ for (int i = 0 ; i < col . size (); i ++){ for (auto it : adj [i]){ if (i != it && col [i] == col [it]) return false ; } } return true ; } bool genratecolor (int i , vector < int > col , int m , vector < int > adj []){ if (i >= col . size ()){ if (goodcolor (adj , col)) return true ; return false ; } for (int j = 0 ; j < m ; j ++){ col [i] = j ; if (genratecolor (i + 1 , col , m , adj)) return true ; col [i] = -1 ; } return false ; } bool graphColoring (int v , vector < vector < int >> & edges , int m) { vector < int > adj [v]; // Build adjacency list from edges for (auto it : edges) { adj [it [0]]. push_back (it [1]); adj [it [1]]. push_back (it [0]); } vector < int > color (v , -1); return genratecolor (0 , color , m , adj); } int main () { int V = 4 ; vector < vector < int >> edges = {{ 0 , 1 }, { 0 , 2 },{ 0 , 3 }, { 1 , 3 }, { 2 , 3 }}; int m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color cout << (graphColoring (V , edges , m) ? "true" : "false") << endl ; return 0 ; } Java import java.util.* ; class GfG { static boolean goodcolor (List < Integer > adj [] , int [] col) { for (int i = 0 ; i < col . length ; i ++) { for (int it : adj [i]) { if (i != it && col [i] == col [it]) return false ; } } return true ; } static boolean genratecolor (int i , int [] col , int m , List < Integer > adj []) { if (i >= col . length) { if (goodcolor (adj , col)) return true ; return false ; } for (int j = 0 ; j < m ; j ++) { col [i] = j ; if (genratecolor (i + 1 , col , m , adj)) return true ; col [i] = - 1 ; } return false ; } static boolean graphColoring (int v , int [][] edges , int m) { List < Integer >[] adj = new ArrayList [v] ; // Build adjacency list from edges for (int i = 0 ; i < v ; i ++) { adj [i] = new ArrayList <> (); } for (int [] it : edges) { adj [it [0]] . add (it [1]); adj [it [1]] . add (it [0]); } int [] color = new int [v] ; Arrays . fill (color , - 1); return genratecolor (0 , color , m , adj); } public static void main (String [] args) { int V = 4 ; int [][] edges = {{ 0 , 1 }, { 0 , 2 },{ 0 , 3 }, { 1 , 3 }, { 2 , 3 }}; int m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color System . out . println (graphColoring (V , edges , m) ? "true" : "false"); } } Python def goodcolor (adj , col): # Check if the coloring is valid for i in range (len (col)): for it in adj [i]: if i != it and col [i] == col [it]: return False return True def genratecolor (i , col , m , adj): if i >= len (col): if goodcolor (adj , col): return True return False for j in range (m): col [i] = j if genratecolor (i + 1 , col , m , adj): return True col [i] = - 1 return False def graphColoring (v , edges , m): adj = [[] for _ in range (v)] # Build adjacency list from edges for u , w in edges : adj [u] . append (w) adj [w] . append (u) color = [- 1] * v return genratecolor (0 , color , m , adj) # Test V = 4 edges = [[0 , 1], [0 , 2], [0 , 3], [

1 , 3], [2 , 3]] m = 3 # Check if the graph can be colored with m colors # such that no adjacent nodes share the same color print ("true" if graphColoring (V , edges , m) else "false") C# using System ; using System.Collections.Generic ; class GfG { static bool goodcolor (List < int > [] adj , int [] col){ for (int i = 0 ; i < col . Length ; i ++){ foreach (int it in adj [i]){ if (i != it && col [i] == col [it]) return false ; } } return true ; } static bool genratecolor (int i , int [] col , int m , List < int > [] adj){ if (i >= col . Length){ if (goodcolor (adj , col)) return true ; return false ; } for (int j = 0 ; j < m ; j ++){ col [i] = j ; if (genratecolor (i + 1 , col , m , adj)) return true ; col [i] = - 1 ; } return false ; } static bool graphColoring (int v , int [,] edges , int m){ List < int > [] adj = new List < int > [v]; // Build adjacency list from edges for (int i = 0 ; i < v ; i ++){ adj [i] = new List < int > (); } for (int i = 0 ; i < edges . GetLength (0); i ++) { int u = edges [i , 0]; int v2 = edges [i , 1]; adj [u]. Add (v2); adj [v2]. Add (u); } int [] color = new int [v]; for (int i = 0 ; i < v ; i ++) color [i] = - 1 ; return genratecolor (0 , color , m , adj); } public static void Main (string [] args) { int V = 4 ; int [,] edges = {{ 0 , 1 }, { 0 , 2 },{ 0 , 3 }, { 1 , 3 }, { 2 , 3 }}; int m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color Console . WriteLine (graphColoring (V , edges , m) ? "true" : "false"); } } JavaScript function goodcolor (adj , col) { // Check if the coloring is valid for (let i = 0 ; i < col . length ; i ++) { for (let it of adj [i]) { if (i !== it && col [i] === col [it]) return false ; } } return true ; } function genratecolor (i , col , m , adj) { if (i >= col . length) { if (goodcolor (adj , col)) return true ; return false ; } for (let j = 0 ; j < m ; j ++) { col [i] = j ; if (genratecolor (i + 1 , col , m , adj)) return true ; col [i] = - 1 ; } return false ; } function graphColoring (v , edges , m) { let adj = Array . from ({ length : v }, () => []); // Build adjacency list from edges for (let [u , w] of edges) { adj [u]. push (w); adj [w]. push (u); } let color = new Array (v). fill (- 1); return genratecolor (0 , color , m , adj); } // Test let V = 4 ; let edges = [[0 , 1], [0 , 2], [0 , 3], [1 , 3], [2 , 3]]; let m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color console . log (graphColoring (V , edges , m) ? "true" : "false"); Output true Approach 2: Using Backtracking - O(V * m^V) Time and O(V+E) Space Assign colors one by one to different vertices, starting from vertex 0. Before assigning a color, check for safety by considering already assigned colors to the adjacent vertices i.e check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false Illustration: C++ #include <bits/stdc++.h> using namespace std ; // Function to check if it's safe to color the current vertex // with the given color bool issafe (int vertex , int col , vector < int > adj [], vector < int > & color) { for (auto it : adj [vertex]) { // If adjacent vertex has the same color, not safe if (color [it] != -1 && col == color [it]) return false ; } return true ; } // Recursive function to try all colorings bool cancolor (int vertex , int m , vector < int > adj [], vector < int > & color) { // If all vertices are colored successfully if (vertex == color . size ()) return true ; // Try all colors from 0 to m-1 for (int i = 0 ; i < m ; i ++) { if (issafe (vertex , i , adj , color)) { color [vertex] = i ; if (cancolor (vertex + 1 , m , adj , color)) // If the rest can be colored, return true return true ; color [vertex] = -1 ; } } // No valid coloring found return false ; } bool graphColoring (int v , vector < vector < int >> & edges , int m) { vector < int > adj [v]; // Build adjacency list from edges for (auto it : edges) { adj [it [0]]. push_back (it [1]); adj [it [1]]. push_back (it [0]); } vector < int > color (v , -1); return cancolor (0 , m , adj , color); } int main () { int V = 4 ; vector < vector < int >> edges = {{ 0 , 1 }, { 0 , 2 },{ 0 , 3 }, { 1 , 3 }, { 2 , 3 }}; int m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color cout << (graphColoring (V , edges , m) ? "true" : "false") << endl ; return 0 ; } Java import java.util.* ; public class Main { // Function to check if it's safe to color the current // vertex with the given color static boolean issafe (int vertex , int col , List < Integer >[] adj , int [] color) { for (int it : adj [vertex]) { // If adjacent vertex has the same color, not safe if (color [it] != - 1 && col == color [it]) return false ; } return true ; } // Recursive function to try all colorings static boolean cancolor (int vertex , int m , List < Integer >[] adj , int [] color) { // If all vertices are colored successfully if (vertex == color . length) return true ; // Try all colors from 0 to m-1 for (int i = 0 ; i < m ; i ++) { if (issafe (vertex , i , adj , color)) { color [vertex] = i ; if (cancolor (vertex + 1 , m , adj , color)) // If the rest can be colored, return true return true ; color [vertex] = - 1 ; } } return false ; // No valid coloring found } static boolean graphColoring (int v , int [][] edges , int m) { List < Integer >[] adj = new ArrayList [v] ; for (int i = 0 ; i < v ; i ++) adj [i] = new ArrayList <> (); // Build adjacency list from edges for (int [] it : edges) { adj [it [0]] . add (it [1]); adj [it [1]] . add (it [0]); } int [] color = new int [v] ; Arrays . fill (color , - 1); return cancolor (0 , m , adj , color); } public static void main (String [] args) { int V = 4 ; int [][] edges = {{ 0 , 1 }, { 0 , 2 },{ 0 , 3 }, { 1 , 3 }, { 2 , 3 }}; int m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color System . out . println (graphColoring (V , edges , m) ? "true" : "false"); } } Python # Function to check if it's safe to color the current vertex # with the given color def issafe (

vertex , col , adj , color): for it in adj [vertex]: # If adjacent vertex has the same color, not safe if color [it] != - 1 and col == color [it]: return False return True # Recursive function to try all colorings def cancolor (vertex , m , adj , color): # If all vertices are colored successfully if vertex == len (color): return True # Try all colors from 0 to m-1 for i in range (m): if issafe (vertex , i , adj , color): color [vertex] = i if cancolor (vertex + 1 , m , adj , color): # If the rest can be colored, return true return True color [vertex] = - 1 # Backtrack # No valid coloring found return False # Main function to set up the graph and call coloring logic def graphColoring (v , edges , m): adj = [[] for _ in range (v)] # Build adjacency list from edges for u , w in edges : adj [u] . append (w) adj [w] . append (u) color = [- 1] * v return cancolor (0 , m , adj , color) # Driver code if __name__ == "__main__" : V = 4 edges = [[0 , 1], [0 , 2], [0 , 3], [1 , 3], [2 , 3]] m = 3 # Check if the graph can be colored with m colors # such that no adjacent nodes share the same color print ("true" if graphColoring (V , edges , m) else "false") C# using System ; using System.Collections.Generic ; class GfG { // Function to check if it's safe to color the current // vertex with the given color static bool issafe (int vertex , int col , List < int > [] adj , int [] color){ foreach (int it in adj [vertex]){ // If adjacent vertex has the same color, not safe if (color [it] != - 1 && col == color [it]) return false ; } return true ; } // Recursive function to try all colorings static bool cancolor (int vertex , int m , List < int > [] adj , int [] color){ // If all vertices are colored successfully if (vertex == color . Length) return true ; // Try all colors from 0 to m-1 for (int i = 0 ; i < m ; i ++){ if (issafe (vertex , i , adj , color)){ color [vertex] = i ; if (cancolor (vertex + 1 , m , adj , color)) // If the rest can be colored, return true return true ; color [vertex] = - 1 ; } } return false ; // No valid coloring found } static bool graphColoring (int v , int [,] edges , int m){ List < int > [] adj = new List < int > [v]; for (int i = 0 ; i < v ; i ++) adj [i] = new List < int > (); // Build adjacency list from edges for (int i = 0 ; i < edges . GetLength (0); i ++){ int u = edges [i , 0]; int v2 = edges [i , 1]; adj [u]. Add (v2); adj [v2]. Add (u); } int [] color = new int [v]; for (int i = 0 ; i < v ; i ++) color [i] = - 1 ; return cancolor (0 , m , adj , color); } static void Main (string [] args){ int V = 4 ; int [,] edges = {{ 0 , 1 }, { 0 , 2 },{ 0 , 3 }, { 1 , 3 }, { 2 , 3 }}; int m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color Console . WriteLine (graphColoring (V , edges , m) ? "true" : "false"); } } JavaScript // Function to check if it's safe to color the current vertex // with the given color function issafe (vertex , col , adj , color) { for (let it of adj [vertex]) { // If adjacent vertex has the same color, not safe if (color [it] !== - 1 && col === color [it]) return false ; } return true ; } // Recursive function to try all colorings function cancolor (vertex , m , adj , color) { // If all vertices are colored successfully if (vertex === color . length) return true ; // Try all colors from 0 to m-1 for (let i = 0 ; i < m ; i ++) { if (issafe (vertex , i , adj , color)) { color [vertex] = i ; if (cancolor (vertex + 1 , m , adj , color)) // If the rest can be colored, return true return true ; color [vertex] = - 1 ; } } // No valid coloring found return false ; } // Main function to set up the graph and call coloring logic function graphColoring (v , edges , m) { let adj = new Array (v). fill (0). map (() => []); // Build adjacency list from edges for (let [u , w] of edges) { adj [u]. push (w); adj [w]. push (u); } let color = new Array (v). fill (- 1); return cancolor (0 , m , adj , color); } // Driver code const V = 4 ; const edges = [[0 , 1], [0 , 2], [0 , 3], [1 , 3], [2 , 3]]; const m = 3 ; // Check if the graph can be colored with m colors // such that no adjacent nodes share the same color console . log (graphColoring (V , edges , m) ? "true" : "false"); Output true Time Complexity: O(V * m V). There is a total of O(m V) combinations of colors. For each attempted coloring of a vertex you call issafe() , can have up to V–1 neighbors, so issafe() is O(V) Auxiliary Space: O(V + E). The recursive Stack of the graph coloring function will require O(V) space, Adjacency list and color array will required O(V+E). Comment Article Tags: Article Tags: Graph Backtracking DSA Samsung Graph Coloring + 1 More