Algorithm:

1. Create a recursive function that takes current index, number of vertices and output color array.
2. If the current index is equal to number of vertices. Check if the output color configuration is safe, i.e check if the adjacent vertices does not have same color. If the conditions are met, print the configuration and break.
3. Assign color to a vertex (1 to m).
4. For every assigned color recursively call the function with next index and number of vertices
5. If any recursive function returns true break the loop and return true.

```
#include <stdbool.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 4

void printSolution(int color[]);

// check if the colored
// graph is safe or not
bool isSafe(
    bool graph[V][V], int color[])
{
    // check for every edge
    for (int i = 0; i < V; i++)
        for (int j = i + 1; j < V; j++)
            if (
                graph[i][j] && color[j] == color[i])
                return false;
    return true;
}

/* This function solves the m Coloring
   problem using recursion. It returns
  false if the m colours cannot be assigned,
  otherwise, return true and prints
  assignments of colours to all vertices.
  Please note that there may be more than
  one solutions, this function prints one
  of the feasible solutions.*/
bool graphColoring(
    bool graph[V][V], int m,
    int i, int color[V])
{
    // if current index reached end
    if (i == V) {
        // if coloring is safe
        if (isSafe(graph, color)) {
            // Print the solution
            printSolution(color);
            return true;
        }
        return false;
    }

    // Assign each color from 1 to m
    for (int j = 1; j <= m; j++) {
        color[i] = j;

        // Recur of the rest vertices
        if (graphColoring(
                graph, m, i + 1, color))
            return true;

        color[i] = 0;
    }

    return false;
}

/* A utility function to print solution */
void printSolution(int color[])
{
    printf(
        "Solution Exists:"
        " Following are the assigned colors \n");
    for (int i = 0; i < V; i++)
        printf(" %d ", color[i]);
    printf("\n");
}

// Driver program to test above function
int main()
{
    /* Create following graph and
       test whether it is 3 colorable
      (3)---(2)
       |   / |
       |  /  |
       | /   |
      (0)---(1)
    */
    bool graph[V][V] = {
        { 0, 1, 1, 1 },
```

```
        { 1, 0, 1, 0 },
    };
    int m = 3; // Number of colors

    // Initialize all color values as 0.
    // This initialization is needed
    // correct functioning of isSafe()
    int color[V];
    for (int i = 0; i < V; i++)
        color[i] = 0;

    if (!graphColoring(
            graph, m, 0, color))
        printf("Solution does not exist");

    return 0;
}
```

Output:

```
Solution Exists: Following are the assigned colors
 1  2  3  2
```

Complexity Analysis:

- **Time Complexity:** O(m^V).
  There are total O(m^V) combination of colors. So the time complexity is O(m^V).
- **Space Complexity:** O(V).
  To store the output array O(V) space is required.

Method 2: Backtracking.

Approach: The idea is to assign colors one by one to different vertices, starting from the 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.

Algorithm:

1. Create a recursive function that takes the graph, current index, number of vertices and output color array.
2. If the current index is equal to number of vertices. Print the color configuration in output array.
3. Assign color to a vertex (1 to m).
4. For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices do not have the same color) recursively call the function with next index and number of vertices
5. If any recursive function returns true break the loop and return true.
6. If no recusive function returns true then return false.

C/C++

```
#include <stdbool.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 4

void printSolution(int color[]);

/* A utility function to check if
   the current color assignment
   is safe for vertex v i.e. checks
   whether the edge exists or not
   (i.e, graph[v][i]==1). If exist
   then checks whether the color to
   be filled in the new vertex(c is
   sent in the parameter) is already
   used by its adjacent
   vertices(i-->adj vertices) or
   not (i.e, color[i]==c) */
bool isSafe(
    int v, bool graph[V][V],
    int color[], int c)
{
    for (int i = 0; i < V; i++)
        if (
            graph[v][i] && c == color[i])
            return false;
    return true;
}

/* A recursive utility function
to solve m coloring problem */
bool graphColoringUtil(
    bool graph[V][V], int m,
    int color[], int v)
{
    /* base case: If all vertices are
       assigned a color then return true */
    if (v == V)
        return true;

    /* Consider this vertex v and
       try different colors */
    for (int c = 1; c <= m; c++) {
        /* Check if assignment of color
           c to v is fine*/
        if (isSafe(
                v, graph, color, c)) {
            color[v] = c;

            /* recur to assign colors to
               rest of the vertices */
            if (
                graphColoringUtil(
                    graph, m, color, v + 1)
                == true)
                return true;

            /* If assigning color c doesn't
               lead to a solution then remove it */
            color[v] = 0;
        }
    }

    /* If no color can be assigned to
       this vertex then return false */
```

m Coloring Problem | Backtracking-5

Last Updated: 19-05-2020

Given an undirected graph and a number m, determine if the graph can be coloured with at most m colours such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices.

Input-Output format:

Input:

1. A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.
2. An integer m which is the maximum number of colors that can be used.

Output:

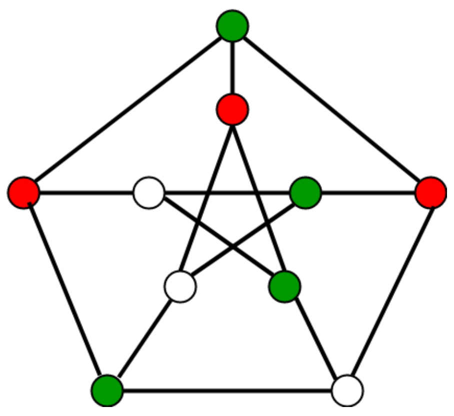
An array color[V] that should have numbers from 1 to m. color[i] should represent the color assigned to the ith vertex. The code should also return false if the graph cannot be colored with m colors.

Example:

```
Input:
graph = {0, 1, 1, 1},
        {1, 0, 1, 0},
        {1, 1, 0, 1},
        {1, 0, 1, 0}
Output:
Solution Exists:
Following are the assigned colors
 1  2  3  2
Explanation: By coloring the vertices
with following colors, adjacent
vertices does not have same colors

Input:
graph = {1, 1, 1, 1},
        {1, 1, 1, 1},
        {1, 1, 1, 1},
        {1, 1, 1, 1}
Output: Solution does not exist.
Explanation: No solution exits.
```

Following is an example of a graph that can be coloured with 3 different colours.



We strongly recommend that you click here and practice it, before moving on to the solution.

Method 1: Naive.

Naive Approach: Generate all possible configurations of colours. Since each node can be coloured using any of the m available colours, the total number of colour configurations possible are m^V.

After generating a configuration of colour, check if the adjacent vertices have the same colour or not. If the conditions are met, print the combination and break the loop.

```
/* This function solves the m Coloring
   problem using Backtracking. It mainly
   uses graphColoringUtil() to solve the
   problem. It returns false if the m
   colors cannot be assigned, otherwise
   return true and prints assignments of
   colors to all vertices. Please note
   that there may be more than one solutions,
   this function prints one of the
   feasible solutions.*/
bool graphColoring(
    bool graph[V][V], int m)
{
    // Initialize all color values as 0.
    // This initialization is needed
    // correct functioning of isSafe()
    int color[V];
    for (int i = 0; i < V; i++)
        color[i] = 0;

    // Call graphColoringUtil() for vertex 0
    if (
        graphColoringUtil(
            graph, m, color, 0)
        == false) {
        printf("Solution does not exist");
        return false;
    }

    // Print the solution
    printSolution(color);
    return true;
}

/* A utility function to print solution */
void printSolution(int color[])
{
    printf(
        "Solution Exists:"
        " Following are the assigned colors \n");
    for (int i = 0; i < V; i++)
        printf(" %d ", color[i]);
    printf("\n");
}

// driver program to test above function
int main()
{
    /* Create following graph and test
       whether it is 3 colorable
      (3)---(2)
       |   / |
       |  /  |
       | /   |
      (0)---(1)
    */
    bool graph[V][V] = {
        { 0, 1, 1, 1 },
        { 1, 0, 1, 0 },
        { 1, 1, 0, 1 },
        { 1, 0, 1, 0 },
    };
    int m = 3; // Number of colors
    graphColoring(graph, m);
    return 0;
}
```

Java

```
/* Java program for solution of
   M Coloring problem using backtracking */
public class mColoringProblem {
    final int V = 4;
    int color[];

    /* A utility function to check
       if the current color assignment
       is safe for vertex v */
    boolean isSafe(
        int v, int graph[][], int color[],
        int c)
    {
        for (int i = 0; i < V; i++)
            if (
                graph[v][i] == 1 && c == color[i])
                return false;
        return true;
    }

    /* A recursive utility function
       to solve m coloring  problem */
    boolean graphColoringUtil(
        int graph[][], int m,
        int color[], int v)
    {
        /* base case: If all vertices are
           assigned a color then return true */
        if (v == V)
            return true;

        /* Consider this vertex v and try
           different colors */
        for (int c = 1; c <= m; c++) {
            /* Check if assignment of color c to v
               is fine*/
            if (isSafe(v, graph, color, c)) {
                color[v] = c;

                /* recur to assign colors to rest
                   of the vertices */
                if (
                    graphColoringUtil(
                        graph, m,
                        color, v + 1))
                    return true;

                /* If assigning color c doesn't lead
```

```
        }

        /* If no color can be assigned to
           this vertex then return false */
        return false;
    }

    /* This function solves the m Coloring problem using
       Backtracking. It mainly uses graphColoringUtil()
       to solve the problem. It returns false if the m
       colors cannot be assigned, otherwise return true
       and  prints assignments of colors to all vertices.
       Please note that there  may be more than one
       solutions, this function prints one of the
       feasible solutions.*/
    boolean graphColoring(int graph[][], int m)
    {
        // Initialize all color values as 0. This
        // initialization is needed correct
        // functioning of isSafe()
        color = new int[V];
        for (int i = 0; i < V; i++)
            color[i] = 0;

        // Call graphColoringUtil() for vertex 0
        if (
            !graphColoringUtil(
                graph, m, color, 0)) {
            System.out.println(
                "Solution does not exist");
            return false;
        }

        // Print the solution
        printSolution(color);
        return true;
    }

    /* A utility function to print solution */
    void printSolution(int color[])
    {
        System.out.println(
            "Solution Exists: Following"
            + " are the assigned colors");
        for (int i = 0; i < V; i++)
            System.out.print(" " + color[i] + " ");
        System.out.println();
    }

    // driver program to test above function
    public static void main(String args[])
    {
        mColoringProblem Coloring
= new mColoringProblem();
        /* Create following graph and
           test whether it is
           3 colorable
          (3)---(2)
           |   / |
           |  /  |
           | /   |
          (0)---(1)
        */
        int graph[][] = {
            { 0, 1, 1, 1 },
            { 1, 0, 1, 0 },
            { 1, 1, 0, 1 },
            { 1, 0, 1, 0 },
        };
        int m = 3; // Number of colors
        Coloring.graphColoring(graph, m);
    }
}
// This code is contributed by Abhishek Shankhadhar
```

Python

```
# Python program for solution of M Coloring
# problem using backtracking

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]\
                              for row in range(vertices)]

    # A utility function to check if the current color assignment
    # is safe for vertex v
    def isSafe(self, v, colour, c):
        for i in range(self.V):
            if self.graph[v][i] == 1 and colour[i] == c:
                return False
        return True

    # A recursive utility function to solve m
    # coloring  problem
    def graphColourUtil(self, m, colour, v):
        if v == self.V:
            return True

        for c in range(1, m + 1):
            if self.isSafe(v, colour, c) == True:
                colour[v] = c
                if self.graphColourUtil(m, colour, v + 1) == True:
                    return True
                colour[v] = 0

    def graphColouring(self, m):
        colour = [0] * self.V
        if self.graphColourUtil(m, colour, 0) == None:
            return False

        # Print the solution
        print "Solution exist and Following are the assigned colours:"
        for c in colour:
            print c,
        return True
```

```
g.graph = [[0, 1, 1, 1], [1, 0, 1, 0], [1, 1, 0, 1], [1, 0, 1, 0]]
m = 3
g.graphColouring(m)

# This code is contributed by Divyanshu Mehta
```

C#

```
/* C# program for solution of M Coloring problem
using backtracking */
using System;

class GFG {
    readonly int V = 4;
    int[] color;

    /* A utility function to check if the current
    color assignment is safe for vertex v */
    bool isSafe(int v, int[, ] graph,
                int[] color, int c)
    {
        for (int i = 0; i < V; i++)
            if (graph[v, i] == 1 && c == color[i])
                return false;
        return true;
    }

    /* A recursive utility function to solve m
    coloring problem */
    bool graphColoringUtil(int[, ] graph, int m,
                           int[] color, int v)
    {
        /* base case: If all vertices are assigned
        a color then return true */
        if (v == V)
            return true;

        /* Consider this vertex v and try different
        colors */
        for (int c = 1; c <= m; c++) {
            /* Check if assignment of color c to v
            is fine*/
            if (isSafe(v, graph, color, c)) {
                color[v] = c;

                /* recur to assign colors to rest
                of the vertices */
                if (graphColoringUtil(graph, m,
                                      color, v + 1))
                    return true;

                /* If assigning color c doesn't lead
                to a solution then remove it */
                color[v] = 0;
            }
        }

        /* If no color can be assigned to this vertex
        then return false */
        return false;
    }

    /* This function solves the m Coloring problem using
    Backtracking. It mainly uses graphColoringUtil()
    to solve the problem. It returns false if the m
    colors cannot be assigned, otherwise return true
    and prints assignments of colors to all vertices.
    Please note that there may be more than one
    solutions, this function prints one of the
    feasible solutions.*/
    bool graphColoring(int[, ] graph, int m)
    {
        // Initialize all color values as 0. This
        // initialization is needed correct functioning
        // of isSafe()
        color = new int[V];
        for (int i = 0; i < V; i++)
            color[i] = 0;

        // Call graphColoringUtil() for vertex 0
        if (!graphColoringUtil(graph, m, color, 0)) {
            Console.WriteLine("Solution does not exist");
            return false;
        }

        // Print the solution
        printSolution(color);
        return true;
    }

    /* A utility function to print solution */
    void printSolution(int[] color)
    {
        Console.WriteLine("Solution Exists: Following"
                          + " are the assigned colors");
        for (int i = 0; i < V; i++)
            Console.Write(" " + color[i] + " ");
        Console.WriteLine();
    }

    // Driver Code
    public static void Main(String[] args)
    {
        GFG Coloring = new GFG();

        /* Create following graph and test whether it is
        3 colorable
        (3)---(2)
        | / |
        | / |
        | / |
        (0)---(1)
        */
        int[, ] graph = { { 0, 1, 1, 1 },
                          { 1, 0, 1, 0 },
                          { 1, 1, 0, 1 },
                          { 1, 0, 1, 0 } };
        int m = 3; // Number of colors
```

```
// This code is contributed by PrinciRaj1992
```

Output:

```
Solution Exists: Following are the assigned colors
 1  2  3  2
```

Complexity Analysis:

- **Time Complexity:** O(m^V).
  There are total O(m^V) combination of colors. So time complexity is O(m^V). The upperbound time complexity remains the same but the average time taken will be less.
- **Space Complexity:** O(V).
  To store the output array O(V) space is required.

References:
http://en.wikipedia.org/wiki/Graph_coloring

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Graph Coloring | Set 1 (Introduction and Applications)
Graph Coloring | Set 2 (Greedy Algorithm)
Coloring a Cycle Graph
Mathematics | Planar Graphs and Graph Coloring
Edge Coloring of a Graph
8 queen problem
Applications of Minimum Spanning Tree Problem
The Knight's tour problem | Backtracking-1
N Queen Problem | Backtracking-3
The Stock Span Problem
Ford-Fulkerson Algorithm for Maximum Flow Problem
Stable Marriage Problem
Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)
Travelling Salesman Problem | Set 2 (Approximate using MST)
Channel Assignment Problem
Snake and Ladder Problem
Vertex Cover Problem | Set 1 (Introduction and Approximate Algorithm)
K Centers Problem | Set 1 (Greedy Approximate Algorithm)
Steiner Tree Problem
Hungarian Algorithm for Assignment Problem | Set 1 (Introduction)

Improved By : SarathChandra1, aakashr79, KasraK, reciever, princiraj1992, more

Article Tags : Backtracking Graph Graph Coloring Samsung

Practice Tags : Samsung Graph Backtracking