Course Name:	Analysis of Algorithms	Semester:	IV
Date of Performance:	04-04-2024	Batch No:	D-2
Faculty Name:		Roll No:	16010122323
Faculty Sign & Date:		Grade/Marks:	

Experiment No: 8

Title: **Graph Coloring.**

Aim and Objective of the Experiment:

Implementation of Graph Colouring Backtracking Algorithm
Objective: To learn the Backtracking strategy of problem solving for Graph Colouring problem

COs to be achieved:

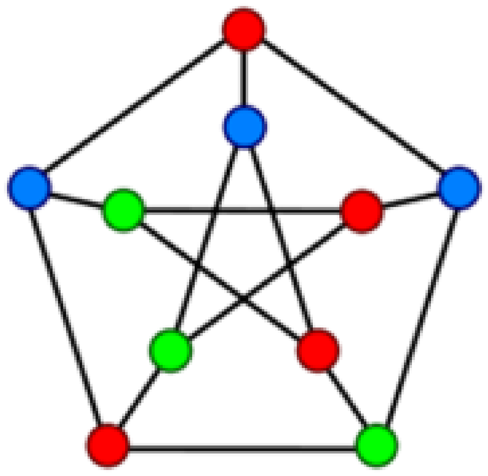
CO2: Describe various algorithm design strategies to solve different problems.

Theory:

Historical Profile:
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. Here coloring of a graph means assignment of colors to all vertices.
Input:

1) A 2D array graph [V][V] where V is the number of vertices in the graph and graph[V][V] is the adjacency matrix representation of the graph.
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.
Following is an example graph that can be colored with 3 colors.



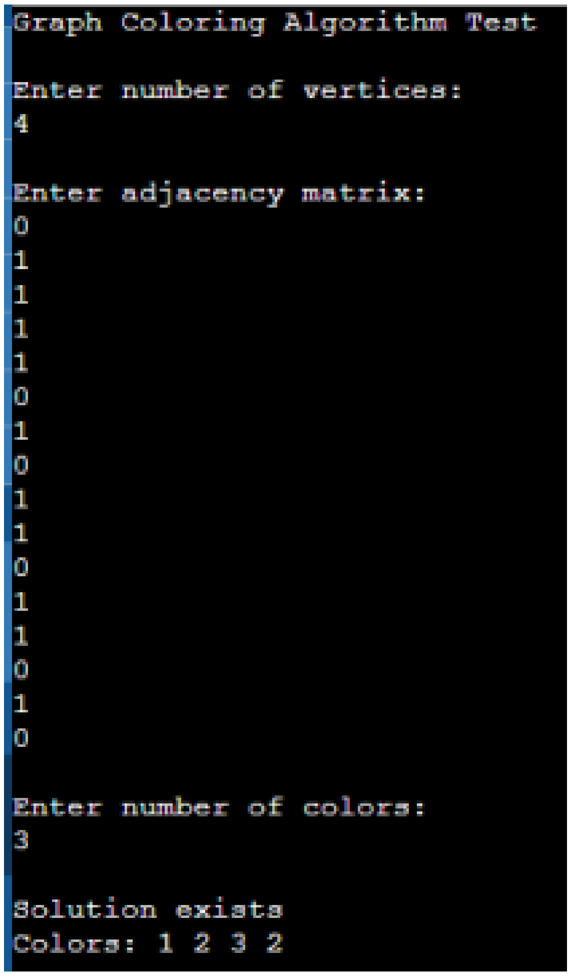
Stepwise-Procedure / Algorithm:

Algorithm Graph Coloring

```
Algorithm mColoring(k)
// This algorithm was formed using the recursive backtracking
// schema. The graph is represented by its boolean adjacency
// matrix G[1 : n, 1 : n]. All assignments of 1, 2, ..., m to the
// vertices of the graph such that adjacent vertices are
// assigned distinct integers are printed. k is the index
// of the next vertex to color.
{
    repeat
    {// Generate all legal assignments for x[k].
        NextValue(k); // Assign to x[k] a legal color.
        if (x[k] = 0) then return; // No new color possible
        if (k = n) then     // At most m colors have been
                            // used to color the n vertices.
            write (x[1 : n]);
        else mColoring(k + 1);
    } until (false);
}
```

Code and output

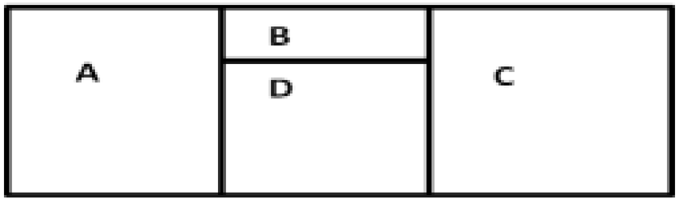
```
import java.util.*;

public class GraphColoring
{
    private int V, numOfColors;
    private int[] color;
    private int[][] graph;

    public void graphColor(int[][] g, int noc)
    {
        V = g.length;
        numOfColors = noc;
        color = new int[V];
        graph = g;

        try
        {
            solve(0);
            System.out.println("No solution");
        }

        catch (Exception e)
        {
            System.out.println("\nSolution exists");
            display();
        }
    }

    public void solve(int v) throws Exception
    {
        if (v == V)
            throw new Exception("Solution found");

        for (int c = 1; c <= numOfColors; c++)
        {
            if (isPossible(v, c))
            {
                color[v] = c;
                solve(v + 1);
                color[v] = 0; // Backtrack if wrong assignment
            }
        }
    }

    public boolean isPossible(int v, int c)
    {
        for (int i = 0; i < V; i++)
        {
            if (graph[v][i] == 1 && c == color[i])
            {
                return false;
            }
        }
        return true;
    }

    public void display()
    {
        System.out.print("Colors: ");
        for (int i = 0; i < V; i++)
        {
            System.out.print(color[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Graph Coloring Algorithm Test\n");

        GraphColoring gc = new GraphColoring();

        System.out.println("Enter number of vertices:");
        int V = scan.nextInt();

        System.out.println("\nEnter adjacency matrix:");
        int[][] graph = new int[V][V];
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                graph[i][j] = scan.nextInt();
            }
        }

        System.out.println("\nEnter number of colors:");
        int c = scan.nextInt();

        gc.graphColor(graph, c);
    }
}
```



```
Graph Coloring Algorithm Test

Enter number of vertices:
4

Enter adjacency matrix:
0
1
1
1
1
0
1
0
1
1
0
1
1
0
1
0

Enter number of colors:
3

Solution exists
Colors: 1 2 3 2
```

Post Lab Subjective/Objective type Questions:

For the given graph, compute the minimum chromatic number to color the graph such that no two adjacent vertices have the same color. Give all possible combinations of such color assignments using backtracking.
Draw state space tree, backtracking tree, solution tree.



Minimum chromatic number: 4