|  |  |
| --- | --- |
| **Course Code: CT2352** | **Course Name: Lab-DAA** |

|  |  |
| --- | --- |
| **Name of Students: S Akshansh** | **Semester/ Section: 6 A** |
| **Roll No: 72** | **Enroll No: 19010927** |

**Practical No 8**

|  |
| --- |
| **Aim**: Write a program to find the minimum of colours required to colour a map of Australia showing each of its states and territories. |
| **Theory:-**  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 coloured with the same colour. Here colouring of a graph means the assignment of colours 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 an 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 is the maximum number of colours that can be used.   **Output:**  An array colour[V] that should have numbers from 1 to m. colour[i] should represent the colour assigned to the ith vertex. The code should also return false if the graph cannot be coloured with m colours.  **Time Complexity: O(m^V).**  There are total O(m^V) combination of colours. So time complexity is O(m^V). The upper bound time complexity remains the same but the average time taken will be less  **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 the number of vertices. Print the color configuration in output array. 3. Assign a 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 recursive function returns true then return false.   **Code:-**  #include<bits/stdc++.h>  using namespace std;  #define V 7  void printSolution(int color[]);  bool isSafe(bool graph[V][V], int color[])  {  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;  }  bool graphColoring(bool graph[V][V], int m, int i,  int color[V])  {  if (i == V) {  if (isSafe(graph, color)) {  printSolution(color);  return true;  }  return false;  }  for (int j = 1; j <= m; j++) {  color[i] = j;  if (graphColoring(graph, m, i + 1, color))  return true;  color[i] = 0;  }  return false;  }  void printSolution(int color[])  {  cout << "Solution Exists:" " Following are the assigned colors \n";  for (int i = 0; i < V; i++)  cout << " " << color[i];  cout << "\n";  }  // Driver code  int main()  {  bool graph[V][V] = {  { 0, 1, 1, 0, 0, 0, 0 },  { 1, 0, 1, 1, 0, 0, 0 },  { 1, 1, 0, 1, 1, 1, 0 },  { 0, 1, 1, 0, 1, 0, 0 },  { 0, 0, 1, 1, 0, 1, 0 },  { 0, 0, 1, 0, 1, 0, 0 },  { 0, 0, 0, 0, 0, 0, 0 },  };  int m = 3;  int color[V];  for (int i = 0; i < V; i++)  color[i] = 0;    if (!graphColoring(graph, m, 0, color))  cout << "Solution does not exist";  cout << "Minimum number of colors required : " << m;  return 0;  }  **Output:-** |
| **Conclusion:** Thus, I have learnt about map coloring technique using backtracking algorithm. |