|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Analysis of Algorithms** | **Semester:** | **IV** |
| **Date of Performance:** | **27 / 03 / 2024** | **Batch No:** | **A – 2** |
| **Faculty Name:** | **Dr. Aarti Phadke** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade / Marks:** | **\_\_\_ / 25** |

**Experiment No.: 8**

**Title: Graph Coloring**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| Implementation of Graph Coloring using 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. |

|  |
| --- |
| **Apparatus / Software Tools Used:** |
| 1. VS Code 2. Microsoft Excel |

|  |
| --- |
| **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:   1. 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:** |
|  |

|  |
| --- |
| **Upload the code / Output:** |
| Code:  def isSafe(graph, color, v, c):      for i in range(len(graph[v])):          if graph[v][i] == 1 and color[i] == c:              return False        return True  def graphColoringUtil(graph, m, color, v):      if v == len(graph):          print(color)          return True        for c in range(1, m + 1):          if isSafe(graph, color, v, c):              color[v] = c                if graphColoringUtil(graph, m, color, v + 1):                  pass              color[v] = 0        return False  def graphColoring(graph, m):      color = [0] \* len(graph)      graphColoringUtil(graph, m, color, 0)  if \_\_name\_\_ == "\_\_main\_\_":      print("\nGraph Coloring Problem\n")        V = int(input("enter number of vertices: "))      graph = []      print("\nenter adjacency matrix:")      for \_ in range(V):          row = list(map(int, input().split()))          graph.append(row)      m = int(input("\nenter the number of colors: "))      graphColoring(graph, m)  Code Output: |

|  |
| --- |
| **Post Lab Subjective / Objective Type Questions:** |
| **For the given graph, compute the minimum chromatic number to colour the graph such that no two adjacent vertices have the same colour. Give all possible combinations of such colour assignments using backtracking.**  **Draw state space tree, backtracking tree, solution tree.**  C:\Users\Admin\Downloads\image.png  Code Output:    Handwritten Solution: |
| **Conclusion:** |
| In conclusion, our experiment on graph coloring using backtracking taught us a practical approach to solving complex optimization challenges efficiently. |

**Signature of faculty in-charge with Date:**