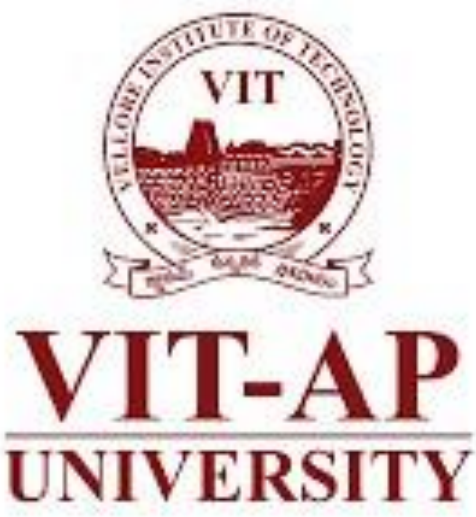

LAB REPORT

Assignment 5

NAME : P.SATHWIKA

REGISTRATION NUMBER : 21BCE8118

SUBJECT : Design and Analysis of Algorithms.

LAB SLOT : L53+L54

IMPLEMENTATION OF GRAPH COLORING

USING BACKTRACKING APPROACH

Graph coloring refers to the problem of coloring vertices of a graph in such a way that no two adjacent vertices have the same color**.** This is also called the vertex coloring problem.

PSEUDOCODE:

```
function graphColoring(graph, colors, vertex):
    if all vertices are colored:
        return true
    for each color in colors:
        if isSafe(graph, vertex, color):
            assign color to vertex
            if graphColoring(graph, colors, vertex + 1) is true:
                return true
            remove color from vertex
    return false


function isSafe(graph, vertex, color):
    for each adjacent_vertex in graph[vertex]:
        if adjacent_vertex is colored with color:
            return false
    return true
```

ALGORITHM:

Start with an empty coloring of the graph.

Pick the first uncolored vertex.

Try to color it with a valid color.

If a valid color is found, move to the next uncolored vertex and repeat step 3.

If no valid color is found for the current vertex, backtrack to the previous vertex and try a different color.

Repeat this process until all vertices are colored or no valid coloring is possible.

CODE:

```
#sathwika 21BCE8118
print("P.Sathwika 21BCE8118\n")
V = 4
def print_solution(color):
    print("Solution Exists: Following are the assigned colors")
    print(" ".join(map(str, color)))
def is_safe(v, graph, color, c):
    for i in range(V):
        if graph[v][i] and c == color[i]:
            return False
    return True
def graph_coloring_util(graph, m, color, v):
    if v == V:
        return True
    for c in range(1, m + 1):
        if is_safe(v, graph, color, c):
            color[v] = c
            if graph_coloring_util(graph, m, color, v + 1):
                return True
            color[v] = 0
    return False
def graph_coloring(graph, m):
    color = [0] * V
    if not graph_coloring_util(graph, m, color, 0):
        print("Solution does not exist")
        return False
```

print_solution(color)

Example usage:

graph = [[0, 1, 1, 0], [1, 0 ,1 ,0], [1 ,0 ,0 ,1], [1 ,0 ,1 ,1]]

graph_coloring(graph, 3)

```python
#sathwika 21BCE8118
print("P.Sathwika 21BCE8118\n")
V = 4
def print_solution(color):
    print("Solution Exists: Following are the assigned colors")
    print(" ".join(map(str, color)))
def is_safe(v, graph, color, c):
    for i in range(V):
        if graph[v][i] and c == color[i]:
            return False
    return True
def graph_coloring_util(graph, m, color, v):
    if v == V:
        return True
    for c in range(1, m + 1):
        if is_safe(v, graph, color, c):
            color[v] = c
            if graph_coloring_util(graph, m, color, v + 1):
                return True
            color[v] = 0
    return False
def graph_coloring(graph, m):
    color = [0] * V
    if not graph_coloring_util(graph, m, color, 0):
        print("Solution does not exist")
        return False
    print_solution(color)
# Example usage:
graph = [[0, 1, 1, 0], [1, 0 ,1 ,0], [1 ,0 ,0 ,1], [1 ,0 ,1 ,1]]
graph_coloring(graph, 3)
```

OUTPUT:

```
P.Sathwika 21BCE8118

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