Date:03.09.25

TASK:6

Solve a Map Coloring problem using constraint satisfaction approach by applying following constraints

AIM

To implement a graph coloring algorithm that assigns colors to university departments in such a way that no two neighboring nodes share the same color, using Python

ALGORITHM

1: Start the program.

2: Represent the departments/districts as a **graph**, where nodes represent departments and edges represent adjacency (neighbors).

3: Define the set of available **colors** (e.g., Red, Green, Blue).

4: Initialize an empty **assignment** to keep track of the color chosen for each department.

5: Create a function is_safe(dept, color) that checks whether assigning a particular color to a department is valid (i.e., no neighbor has the same color).

6: Create a recursive function assign_colors(departments, index) that tries to assign colors to each department.

7: If all departments are assigned a valid color (index == total departments), return success.

8: For the current department, try each available color:

- If safe, assign it temporarily and move to the next department.
- If not safe, try the next color.

9: If no color can be assigned, backtrack by removing the assignment and return failure.

10: After recursion finishes, print the assigned colors if successful, else print that no valid coloring is possible.

PROGRAM

University Departments Decoration

```
# Departments adjacency (Graph as dictionary)
graph = {
   "A": ["B", "C", "D"],
   "B": ["A", "C"],
   "C": ["A", "B", "D"],
   "D": ["A", "C"]
}

# Available theme colors
colors = ["Red", "Green", "Blue"]

# Store final assignment
assignment = {}

# Check if assigning a color is safe
def is_safe(dept, color):
   for neighbor in graph[dept]:
      if neighbor in assignment and assignment[neighbor] == color:
         return False
   return True

# Backtracking function
def assign_colors(departments, index=0):
   if index == len(departments):
      return True

   dept = departments[index]

   for color in colors:
      if is_safe(dept, color):
         assignment[dept] = color
         if assign_colors(departments, index + 1):
```

```
            return True
        assignment.pop(dept)  # backtrack

    return False


# Main
departments = list(graph.keys())

if assign_colors(departments):
    print("✅ Valid theme coloring found:")
    for dept, color in assignment.items():
        print(f"Department {dept} → {color}")
else:
    print("❌ No valid coloring possible.")
```

OUTPUT

Valid theme coloring found:

Department A → Red

Department B → Green

Department C → Blue

Department D → Green

RESULT

Thus, the implementation a graph coloring algorithm that assigns colors to university departments in such a way that no two neighboring nodes share the same color, using Python was successfully executed and output was verified.