

## **TASK 6**

### **Map Coloring**

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

- Assign each territory a color such that no two adjacent territories have the same color by considering following parameters: Domains, Variables and Constraints
- Apply Basic Greedy Coloring Algorithm: Color first vertex with first color, do following for remaining  $V-1$  vertices.
- Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to  $v$ , assign a new color to it.

**Tools- Python, Online Simulator - <https://graphonline.ru/en/>**

#### **PROBLEM STATEMENT:**

**CO3 K3**

In a fictional country, there are several political districts that need to be colored for an upcoming election. Each district is represented as a region on a map, and the goal is to assign a color to each district such that no two neighboring districts have the same color. The constraint satisfaction approach will be used to solve the Map Coloring problem. The country has multiple political parties, and each party has a specific set of colors associated with it. The districts are interconnected, and no two adjacent districts can be assigned the same color since it represents political affiliations. The objective is to ensure that the map is colored in a way that avoids any potential clashes between neighboring districts' colors.

The goal is to ensure a fair and visually appealing political districting map by assigning colors to districts in a way that respects the political affiliations and avoids color clashes between neighboring regions.

## TASK:6

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

To implement a map coloring algorithm that assigns color to political districts in such a way that two adjacent districts share the same color, using Python.

**ALGORITHM**

1. **Start.**  
Represent the map as a graph where each **district** is a node and edges represent **adjacent districts**.
2. **Input.**  
Provide the adjacency list of the graph (district connections) and the set of available colors (party colors).
3. **Initialize.**  
Create a dictionary (color assignment) to store the color chosen for each district, initially set to **None**.
4. **Select a District.**  
Pick the next uncolored district from the list of districts.
5. **Check Colors.**  
For the current district, try assigning each available color from the list.
6. **Verify Safety.**  
For each attempted color, check if any of the **neighboring districts** already has the same color.
  - If yes → reject that color.
  - If no → temporarily assign the color.
7. **Recursive Call.**  
Recursively move to the **next district** and repeat the coloring process.
8. **Backtrack if Needed.**  
If no color is possible for a district, **undo the previous assignment (backtrack)** and try a different color for the previous district.
9. **Check Completion.**  
If all districts have been successfully assigned colors, store/print the solution.
10. **Stop.**  
If a valid coloring exists, output the district–color mapping; otherwise, report that no solution exists.

## PROGRAM

### Map coloring algorithm using CSP that assigns color to political districts

```
# Map Coloring Problem using Constraint Satisfaction (Backtracking)

# Function to check if the current color assignment is valid
def is_safe(graph, color_assignment, district, color):
    for neighbor in graph[district]:
        if color_assignment.get(neighbor) == color:
            return False
    return True

# Backtracking function to assign colors
def assign_colors(graph, colors, color_assignment, district_list, index=0):
    # If all districts are colored, return solution
    if index == len(district_list):
        return True

    district = district_list[index]

    # Try each available color
    for color in colors:
        if is_safe(graph, color_assignment, district, color):
            color_assignment[district] = color

            # Recur to assign colors for the next district
            if assign_colors(graph, colors, color_assignment, district_list, index + 1):
                return True

    # Backtrack
    color_assignment[district] = None

    return False

# Main function
def map_coloring(graph, colors):
    districts = list(graph.keys())
    color_assignment = {district: None for district in districts}

    if assign_colors(graph, colors, color_assignment, districts):
        return color_assignment
    else:
```

```
return None
```

```
# Example: Fictional country with 6 districts
```

```
graph = {  
    "District1": ["District2", "District3"],  
    "District2": ["District1", "District3", "District4"],  
    "District3": ["District1", "District2", "District4", "District5"],  
    "District4": ["District2", "District3", "District5", "District6"],  
    "District5": ["District3", "District4", "District6"],  
    "District6": ["District4", "District5"]  
}
```

```
# Political party colors
```

```
colors = ["Red", "Blue", "Green", "Yellow"]
```

```
solution = map_coloring(graph, colors)
```

```
if solution:
```

```
    print("Map Coloring Solution:")
```

```
    for district, color in solution.items():
```

```
        print(f"{district} -> {color}")
```

```
else:
```

```
    print("No solution found!")
```

## **OUTPUT**

Map Coloring Solution:

District1 -> Red

District2 -> Blue

District3 -> Green

District4 -> Red

District5 -> Blue

District6 -> Green

## **RESULT**

Thus, the implementation a map coloring algorithm that assigns colors to political districts, using Python was successfully executed and output was verified.