

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)

```
graph = {
    "A": ["B", "C", "D"],
    "B": ["A", "C"],
    "C": ["A", "B", "D"],
    "D": ["E", "C"],
    "E": ["A", "B"]
}

color = ["Red", "Green", "Blue"]
assignment = {}

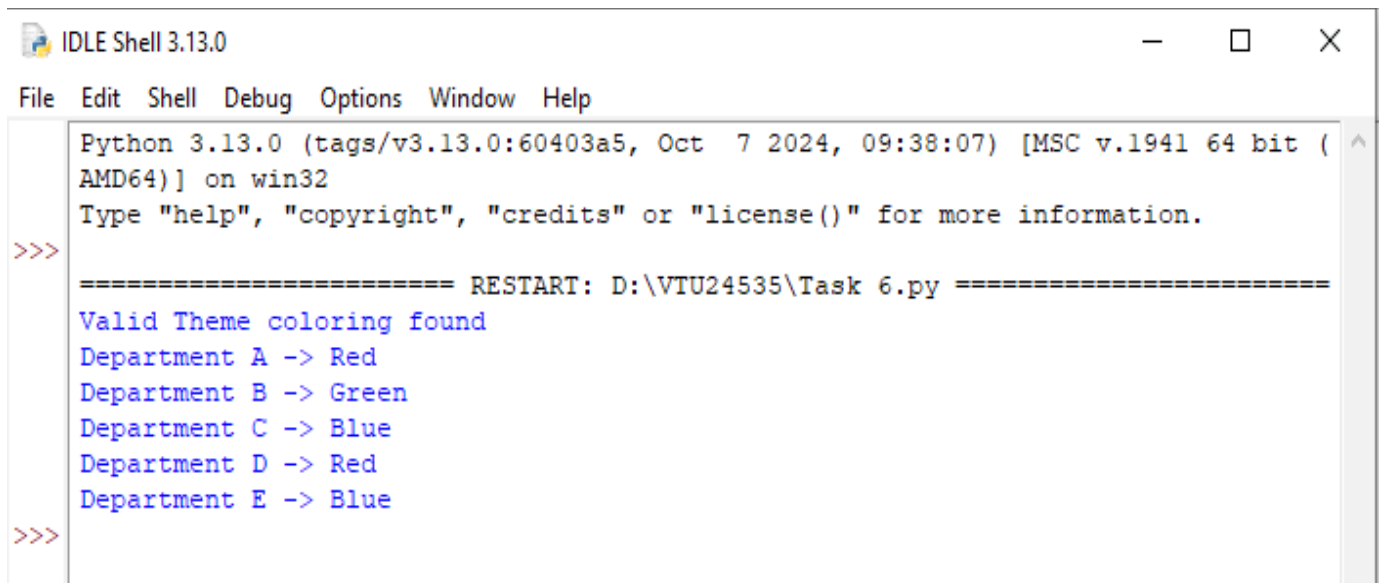
def is_safe(dept, chosen_color):
    for neighbor in graph[dept]:
        if neighbor in assignment and assignment[neighbor] == chosen_color:
            return False
    return True

def assign_color(department, index=0):
    if index == len(department):
        return True

    dept = department[index]
    for chosen_color in color:
        if is_safe(dept, chosen_color):
            assignment[dept] = chosen_color
            if assign_color(department, index + 1):
                return True
            assignment.pop(dept)
    return False

department = list(graph.keys())
if assign_color(department):
    print("Valid Theme coloring found")
    for dept, assigned_color in assignment.items():
        print(f'Department {dept} -> {assigned_color}')
else:
    print("No Valid coloring possible.")
```

OUTPUT

A screenshot of the IDLE Shell 3.13.0 window. The window has a title bar with standard Windows controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area shows the following output:

```
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\VTU24535\Task 6.py =====
Valid Theme coloring found
Department A -> Red
Department B -> Green
Department C -> Blue
Department D -> Red
Department E -> Blue
>>>
```

RESULT

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