11. Write the python program for Map Coloring to implement CSP.

**AIM :** Map Coloring to implement CSP.

**ALGORITHM :**

1. We define the CSP class to encapsulate the problem's variables, domains, and constraints.
2. The ‘is\_consistent’ function checks if a variable assignment is consistent with the constraints.
3. The ‘backtrack’ function recursively attempts to find a consistent assignment.
4. The ‘solve’ method initializes an empty assignment and calls ‘backtrack’ to find a solution.
5. We define variables, domains, and constraints for the map coloring problem.
6. The ‘adjacent\_constraint’ function enforces the constraint that adjacent regions cannot have the same color.
7. We create a CSP instance with the defined data and constraints.
8. We call ‘solve’ to find a solution, and if one is found, we output the colored map.

**PROGRAM :**

def is\_consistent(assignment, variable, value, constraints):

for constraint in constraints:

if variable in constraint['scope']:

related\_var = [v for v in constraint['scope'] if v != variable][0]

if related\_var in assignment and not constraint['func'](value, assignment[related\_var]):

return False

return True

def backtrack(assignment, variables, domains, constraints):

if len(assignment) == len(variables):

return assignment

variable = [var for var in variables if var not in assignment][0]

for value in domains[variable]:

if is\_consistent(assignment.copy(), variable, value, constraints):

assignment[variable] = value

result = backtrack(assignment, variables, domains, constraints)

if result is not None:

return result

assignment.pop(variable)

return None

def map\_coloring(variables, domains, constraints):

assignment = {}

return backtrack(assignment, variables, domains, constraints)

variables = ['WA', 'NT', 'Q', 'NSW', 'V', 'SA', 'T']

domains = {

'WA': ['R', 'G', 'B'],

'NT': ['R', 'G', 'B'],

'Q': ['R', 'G', 'B'],

'NSW': ['R', 'G', 'B'],

'V': ['R', 'G', 'B'],

'SA': ['R', 'G', 'B'],

'T': ['R', 'G', 'B']

}

def different\_colors(color1, color2):

return color1 != color2

constraints = [

{'scope': ['WA', 'NT'], 'func': different\_colors},

{'scope': ['WA', 'SA'], 'func': different\_colors},

{'scope': ['NT', 'SA'], 'func': different\_colors},

{'scope': ['NT', 'Q'], 'func': different\_colors},

{'scope': ['SA', 'Q'], 'func': different\_colors},

{'scope': ['SA', 'NSW'], 'func': different\_colors},

{'scope': ['SA', 'V'], 'func': different\_colors},

{'scope': ['Q', 'NSW'], 'func': different\_colors},

{'scope': ['NSW', 'V'], 'func': different\_colors}

]

solution = map\_coloring(variables, domains, constraints)

if solution:

print("Map Coloring Solution:")

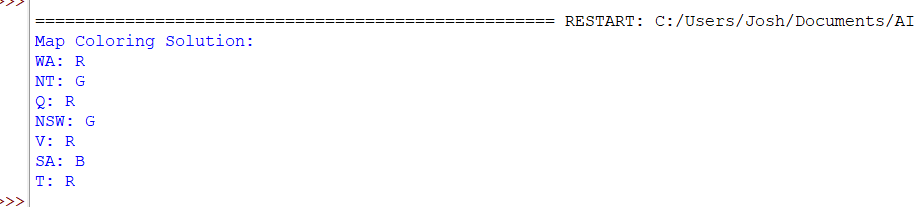
for variable, color in solution.items():

print(f"{variable}: {color}")

else:

print("No solution found.")

**OUT PUT :**

****