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

**Program:**

# Define the map of regions (adjacency list)

regions = {

'WA': ['NT', 'SA'],

'NT': ['WA', 'SA', 'Q'],

'SA': ['WA', 'NT', 'Q', 'NSW', 'V'],

'Q': ['NT', 'SA', 'NSW'],

'NSW': ['Q', 'SA', 'V'],

'V': ['SA', 'NSW'],

'T': [] # Tasmania has no neighbors

}

colors = ['Red', 'Green', 'Blue']

def is\_valid(assignment, region, color):

for neighbor in regions[region]:

if neighbor in assignment and assignment[neighbor] == color:

return False

return True

def select\_unassigned\_variable(assignment, domains):

# MRV heuristic: choose the region with the fewest legal colors

unassigned = [v for v in regions if v not in assignment]

return min(unassigned, key=lambda var: len(domains[var]))

def forward\_checking(region, color, domains):

# Remove chosen color from neighbors’ domains

removed = {}

for neighbor in regions[region]:

if color in domains[neighbor]:

domains[neighbor].remove(color)

removed[neighbor] = color

return removed

def restore\_domains(domains, removed):

# Restore removed colors

for var, color in removed.items():

domains[var].append(color)

def backtrack(assignment, domains):

if len(assignment) == len(regions):

return assignment

var = select\_unassigned\_variable(assignment, domains)

for color in domains[var]:

if is\_valid(assignment, var, color):

assignment[var] = color

removed = forward\_checking(var, color, domains)

result = backtrack(assignment, domains)

if result:

return result

restore\_domains(domains, removed)

del assignment[var]

return None

def map\_coloring():

domains = {region: colors[:] for region in regions}

assignment = {}

result = backtrack(assignment, domains)

return result

# Run the algorithm

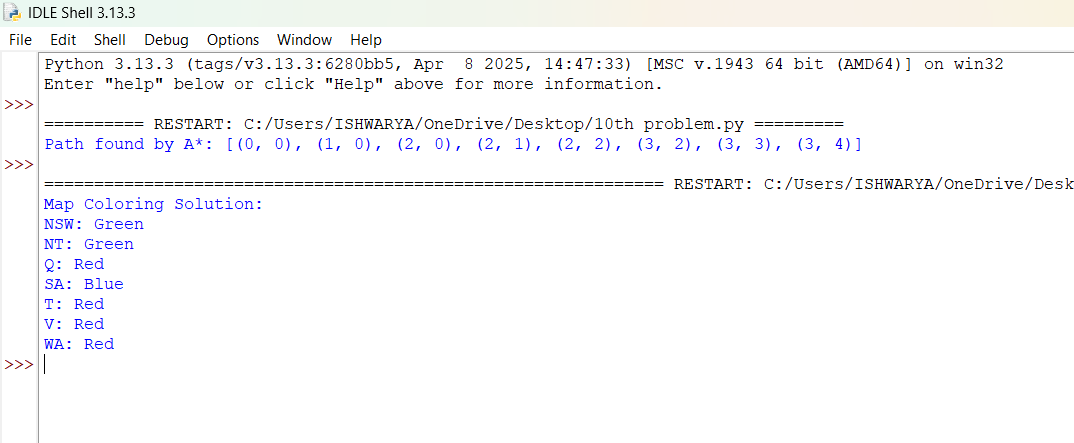
solution = map\_coloring()

print("Map Coloring Solution:")

for region in sorted(solution):

print(f"{region}: {solution[region]}")

**output:**

****