# Map Coloring using Constraint Satisfaction Problem (CSP)

from itertools import product

class MapColoring:

def \_\_init\_\_(self, regions, colors):

self.regions = regions

self.colors = colors

self.adjacency = {region: [] for region in regions}

def add\_edge(self, region1, region2):

self.adjacency[region1].append(region2)

self.adjacency[region2].append(region1)

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

for neighbor in self.adjacency[region]:

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

return False

return True

def backtrack(self, assignment):

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

return assignment

region = self.select\_unassigned\_region(assignment)

for color in self.colors:

if self.is\_valid(region, color, assignment):

assignment[region] = color

result = self.backtrack(assignment)

if result:

return result

del assignment[region]

return None

def select\_unassigned\_region(self, assignment):

for region in self.regions:

if region not in assignment:

return region

return None

def solve(self):

return self.backtrack({})

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

regions = ['A', 'B', 'C', 'D']

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

map\_coloring = MapColoring(regions, colors)

map\_coloring.add\_edge('A', 'B')

map\_coloring.add\_edge('A', 'C')

map\_coloring.add\_edge('B', 'C')

map\_coloring.add\_edge('B', 'D')

map\_coloring.add\_edge('C', 'D')

solution = map\_coloring.solve()

print("Coloring Assignment:", solution)

OUTPUT :

