**COLOR MAPPING PROBLEM**

# CSP Map colouring USing BACKTRACKING

def is\_valid(map, region, color, color\_assignment):

for neighbor in map[region]:

if neighbor in color\_assignment and color\_assignment[neighbor] == color:

return False

return True

def solve\_map\_coloring(map, regions, colors, color\_assignment={}):

if len(color\_assignment) == len(regions):

return color\_assignment

current\_region = [r for r in regions if r not in color\_assignment][0]

for color in colors:

if is\_valid(map, current\_region, color, color\_assignment):

color\_assignment[current\_region] = color

result = solve\_map\_coloring(map, regions, colors, color\_assignment)

if result is not None:

return result

del color\_assignment[current\_region]

return None

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

map = {

"WA": ["NT", "SA"],

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

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

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

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

"V": ["SA", "NSW"],

}

regions = list(map.keys())

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

coloring = solve\_map\_coloring(map, regions, colors)

if coloring:

print("Valid coloring:")

for region, color in coloring.items():

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

else:

print("No valid coloring found.")

**POSSIBLE COLOR MAPPING PROBLEM**

# R is a set of restrictions

# this functions colors the given province with the given color

# returns false if not possible, returns the set of new restrictions if possible

def addColor(R, province, color):

ans = []

for rr in R:

res = checkRestriction(rr, province, color)

if res == False:

return False

elif res == None:

continue

else:

ans.append(res)

return ans

# checks if the restrition rr allows the given province to have the given color

# returns false if not possible, otherwise returns the new restriction

def checkRestriction(rr, province, color):

#finding the index of the province (saved to index)

index = -1

other = -1

if rr[0] == province:

index = 0

other = 1

elif rr[1] == province:

index = 1

other = 0

else:

return rr

if isinstance(rr[other], int):

# other component is a color

if (color != rr[other]):

return None

else:

return False

else:

return [rr[other], color]

# solving the CSP by variable elimination

# recursive structure: ci is the province index to be colored (0 = bc, 1 = ab, etc)

# n is the number of colors

# provinces is a list of provinces

# if coloring is possible returns the province-> color map, otherwise False

def solveCSP(provinces, n, R, ci):

if (ci == 0):

# in the beginning any color can be assigned to the first province, lets say 1

newR = addColor(R, provinces[0], 1)

if (newR == False):

return False

ans = {provinces[0]:1}

res = solveCSP(provinces, n, newR, 1)

if (res == False):

return False

ans.update(res)

return ans

elif (ci == len(provinces)):

return {}

# branching over all possible colors for provinces[ci]

for color in range (1,n+1):

ans = {provinces[ci]:color}

newR = addColor(R, provinces[ci], color)

if (newR == False):

continue

res = solveCSP(provinces, n, newR, ci+1)

if (res == False):

continue

#print(ans)

#print(res)

#print("============")

ans.update(res)

return ans

# no choice for the current province

return False

# main program starts

# ===================================================

n=5 #int(input("Enter the number of color"))

colors=[]

for i in range(1,n+1):

colors.append(i)

#print(colors)

# creating map of canada

# cmap[x] gives the neighbors of the province x

cmap = {}

cmap["ab"] = ["bc","nt","sk"]

cmap["bc"] = ["yt", "nt", "ab"]

cmap["mb"] = ["sk","nu","on"]

cmap["nb"] = ["qc", "ns", "pe"]

cmap["ns"] = ["nb", "pe"]

cmap["nl"] = ["qc"]

cmap["nt"] = ["bc", "yt", "ab", "sk", "nu"]

cmap["nu"] = ["nt", "mb"]

cmap["on"] = ["mb", "qc"]

cmap["pe"] = ["nb", "ns"]

cmap["qc"] = ["on", "nb", "nl"]

cmap["sk"] = ["ab", "mb", "nt"]

cmap["yt"] = ["bc", "nt"]

# CSP restrictions

# each restriction is modeled as a pair [a,b] which means the province a's

# color is not equal to b, where b is either a color (a number 1 to n) or

# another province. Examples ['bc', 'ab'] means the color of bc should

# not be equal to ab -- ["bc",4] means the color of bc should not be 4

# R is the list of restrictions

R = []

# initiaitiong restrictions based on the province neighborhood

for x in cmap:

for y in cmap[x]:

R.append([x,y])

# initiating a list of provinces

provinces = []

for p in cmap:

provinces.append(p)

#print(solveCSP(provinces, 3, R, 0))

while(1):

num=int(input("Enter number of the color? "))

print(solveCSP(provinces, num, R, 0))

#print(R)

#print(" ================= ")

#print(checkRestriction(["ab",4],"ab",4))

#R = addColor(R, 'bc', 4)

#print(R)

#print(" ================= ")

#print(checkRestriction(["ab",4],"ab",4))

#R = addColor(R, 'ab', 4)

#print(R)

**COLORMAP PLOT**

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.colors import ListedColormap

x = np.random.rand(50)

y = np.random.rand(50)

colors = np.random.rand(50)

cmap = ListedColormap(['red', 'green', 'blue'])

plt.scatter(x, y, c=colors, cmap=cmap)

plt.colorbar()

plt.show()

**CUSTOM COLORMAP PLOT**

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.colors import ListedColormap

colors = ["blue", "green", "yellow"]

cmap = ListedColormap(colors)

data = np.random.randn(10, 10)

plt.imshow(data, cmap=cmap)

plt.colorbar()

plt.show()