

**BackTracking\_MapColoring and BackTracking\_8queens**

**ABUBAKAR ASIF**

**#abubakarasif3111@gmail.com**

**#https://github.com/Abubakar3111**

**#https://www.linkedin.com/in/abubakar-asif-b3b94021a/**

**Artificial Intelligence**

**#BackTracking\_MapColoring:-**

#AbubakarAsif FA20-BCE-013

print("AbubakarAsif FA20-BCE-013")

class CSP():

def \_\_init\_\_(self, variables, domains,constraints):

self.variables = variables # variables to be constrained

self.domains = domains # domain of each variable

self.constraints = constraints

# if variable not in self.domains:

# raise LookupError(

# "Every variable should have a domain assigned to it.")

defadd\_constraint(self, constraint):

#pass

#for each variable involved in the contstraint

#append the contraint to the variable's contraint list in the dictionary

for v in constraint.variables:

self.constraints[v].append(constraint)

defconsistent(self, variable, assignment):

#pass

#Check if the value assignment is consistent by checking all constraints

#for the given variable against it

#for each contraint of the variable in the contraints dictionary

#if constraint not statisfied return false

for c in self.constraints[variable]:

if not c.satisfied(assignment):

return False

return True

#otherwise return true

defbacktracking\_search(self, assignment={}):

#pass

# if assignment is complete if every variable is assigned (our base case) the return it

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

return assignment

# get all variables in the CSP but not in the assignment

unassigned = [v for v in self.variables if v not in assignment]

first = unassigned[0]

# for every possible domain value of the first unassigned variable

for value in self.domains[first]:

local\_assignment = assignment.copy()

local\_assignment[first] = value

print(local\_assignment)

if self.consistent(first,local\_assignment):

result = self.backtracking\_search(local\_assignment)

if result != None:

return result

# if we're still consistent, we recurse (continue)

# if we didn't find the result, we will end up backtracking

# if resutl is not none we will return it

class MapColoringConstraint():

def \_\_init\_\_(self, place1, place2):

self.variables = [place1, place2]

def \_\_repr\_\_(self):

return repr(self.variables)

defsatisfied(self, assignment):

#pass

# write code here

# If either place is not in the assignment then satisfied

if self.variables[0] not in assignment or self.variables[1] not in assignment:

return True

if assignment[self.variables[0]] == assignment[self.variables[1]]:

return False

else: return True

# otherwise return if their colors coflicting or not

# check the color assigned to place1 is not the same as the

# color assigned to place2

#Main block of code

#create variables list

variables = ['Western Australia', 'Northern Territory', 'South Australia', 'Queensland', 'New South Wales', 'Victoria', 'Tasmania']

#create domains dictionary {variable1:[domain1],...}

domains = {}

for v in variables:

domains[v] = ['R','G','B']

constraints={}

for v in variables:

constraints[v] = []

#create csp object

csp = CSP(variables,domains,constraints)

#add contraints using add\_constraint method

c1 = MapColoringConstraint("Western Australia", "Northern Territory")

csp.add\_constraint(c1)

# #{'Western Australia': [['Western Australia', 'Northern Territory']],

# #'Northern Territory': [['Western Australia', 'Northern Territory']]} Actually the full constraint object

csp.add\_constraint(MapColoringConstraint("Western Australia", "South Australia"))

csp.add\_constraint(MapColoringConstraint("South Australia", "Northern Territory"))

csp.add\_constraint(MapColoringConstraint("Queensland", "Northern Territory"))

csp.add\_constraint(MapColoringConstraint("Queensland", "South Australia"))

csp.add\_constraint(MapColoringConstraint("Queensland", "New South Wales"))

csp.add\_constraint(MapColoringConstraint("New South Wales", "South Australia"))

csp.add\_constraint(MapColoringConstraint("Victoria", "South Australia"))

csp.add\_constraint(MapColoringConstraint("Victoria", "New South Wales"))

csp.add\_constraint(MapColoringConstraint("Victoria", "Tasmania"))

#print(csp.constraints)

#call backtracking\_search method

assignment= {}

solution = csp.backtracking\_search(assignment)

# print solution if found

if solution != None:

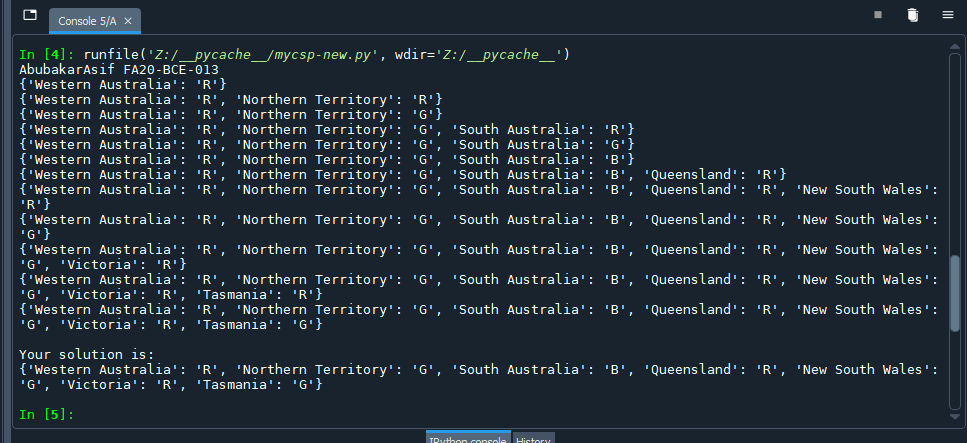
print("\nYour solution is:")

print(solution)

else: print ("No solution found")

print("AbubakarAsif FA20-BCE-013")

# print solution if found



**#BackTracking\_8Queens:-**

print("AbubakarAsif FA20-BCE-013")

class CSP():

def \_\_init\_\_(self, variables, domains,constraints):

self.variables = variables # variables to be constrained

self.domains = domains # domain of each variable

self.constraints = constraints

# if variable not in self.domains:

# raise LookupError(

# "Every variable should have a domain assigned to it.")

defadd\_constraint(self, constraint):

#pass

#for each variable involved in the contstraint

#append the contraint to the variable's contraint list in the dictionary

for v in self.constraints:

self.constraints[v].append(constraint)

defconsistent(self, variable, assignment):

pass

#Check if the value assignment is consistent by checking all constraints

#for the given variable against it

#for each contraint of the variable in the contraints dictionary

#if constraint not statisfied return false

for c in self.constraints[variable]:

if not c.satisfied(assignment):

return False

return True

#otherwise return true

defbacktracking\_search(self, assignment={}):

#pass

# if assignment is complete if every variable is assigned (our base case) then return it

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

return assignment

# get all variables in the CSP but not in the assignment

unassigned = [v for v in self.variables if v not in assignment]

first = unassigned[0]

for value in self.domains[first]:

local\_assignment=assignment.copy()

local\_assignment[first]=value

if self.consistent(first,local\_assignment):

result=self.backtracking\_search(local\_assignment)

if result != None:

return result

return None

class queen\_constraint:

def satisfied(self,assignment):

for i in assignment:

for j in assignment:

if i != j or j>i:

if assignment[i]==assignment[j]:

return False

if abs(i - j)== abs(assignment[i] - assignment[j]):

return False

return True

defcharconv(char):

number=ord(char)-96

return number

variables=[1,2,3,4,5,6,7,8]

domains={}

for v in variables:

domains[v]=[1,2,3,4,5,6,7,8]

constraints={}

for v in variables:

constraints[v]=[]

csp=CSP(variables,domains,constraints)

csp.add\_constraint(queen\_constraint())

#print(csp.constraints)

solution=csp.backtracking\_search()

if solution != None:

print(solution)

else: print("No Solution found")

print("AbubakarAsif FA20-BCE-013")

