import numpy as np

from numpy import random

def fitness(population, n):

for i in range(len(population)):

ap = 0

list = [[0 for \_ in range(n)] for \_ in range(n)]

for j in range(n):

list[j][population[i][j]] = 1

# horizontally

for j in range(n):

qinrow = 0

for k in range(n):

if (list[k][j] == 1):

qinrow += 1

ap += qinrow \* (qinrow - 1) / 2

# print(ap)

# diag1

l = n

c = 0

while c < n:

qindiag = 0

j = c

for k in range(l):

if (list[j][k] == 1):

qindiag += 1

j += 1

l -= 1

c += 1

ap += qindiag \* (qindiag - 1) / 2

# print(ap)

# diag2

l = n - 1

c = 0

while c < n:

qindiag = 0

j = c

k = l

while k >= c:

if (list[j][k] == 1):

qindiag += 1

k -= 1

j += 1

c += 1

ap += qindiag \* (qindiag - 1) / 2

# print(ap)

# diag3

l = n - 1

c = n - 2

while c >= 0:

qindiag = 0

k = c

for j in range(l):

if (list[j][k] == 1):

qindiag += 1

k -= 1

l -= 1

c -= 1

ap += qindiag \* (qindiag - 1) / 2

# diag4

j = n - 1

c = 1

while j >= 0:

qindiag = 0

for k in range(j):

if (list[k][k + c] == 1):

qindiag += 1

c += 1

ap += qindiag \* (qindiag - 1) / 2

j -= 1

fitness\_array[i] = n \* (n - 1) / 2 - ap

return fitness\_array

def select(population, fit):

''' take input: population and fit

fit contains fitness values of each of the individuals

in the population

return: one individual randomly giving

more weight to ones which have high fitness score'''

# a = [0, 1, 2, 3, 4]

# size = 1

# p = [.31, .29, 0.26, 0.14]

for i in range(len(population)):

if (fit[i] < 20):

if (i > 0 or i == len(population) - 1):

population[i] = population[i - 1]

else:

population[i] = population[i + 1]

return population

def crossover(x, y):

'''take input: 2 parents - x, y.

Generate a random crossover point.

Append first half of x with second

half of y to create the child

returns: a child chromosome'''

point = random.randint(n - 1)

while point < len(x):

temp = x[point]

x[point] = y[point]

y[point] = temp

point += 1

return x, y

def mutate(child):

'''take input: a child

mutates a random

gene into another random gene

returns: mutated child'''

genepointer = random.randint(n - 1)

child[genepointer] = random.randint(n - 1)

return child

def GA(population, n, mutation\_threshold):

'''implement the pseudocode here by

calling the necessary functions- Fitness,

Selection, Crossover and Mutation

print: the max fitness value and the

chromosome that generated it which is ultimately

the solution board'''

genmax = 25000

gen = genmax

while gen > 0:

fit = fitness(population, n)

for i in range(len(population)):

if (fit[i] == n \* (n - 1) / 2):

print("Congrats! fittest child found after", genmax - gen, "generation/s")

print("fitness value is ", fit[i])

print("Fittest child is given below:")

return population[i]

population = select(population, fitness\_array)

i = 0

while i < (len(population)):

population[i], population[i + 1] = crossover(population[i], population[i + 1])

i += 2

for i in range(len(population)):

m = random.rand()

if (m < mutation\_threshold):

population[i] = mutate(population[i])

gen -= 1

print("No fittest child in", genmax, "generations")

fit = fitness(population, n)

maxval = max(fit)

print("Highest fitness=> ",maxval)

print("Most fittest child is given below")

maxfit = fit.index(maxval)

return population[maxfit]

'''for 8 queen problem, n = 8'''

n = 8

'''start\_population denotes how many individuals/chromosomes are there

in the initial population n = 8'''

start\_population = 10

'''if you want you can set mutation\_threshold to a higher value,

to increase the chances of mutation'''

mutation\_threshold = 0.6

'''creating the population with random integers between 0 to 7 inclusive

for n = 8 queen problem'''

population = np.random.randint(0, n, (start\_population, n))

print(population)

fitness\_array = [0] \* start\_population

fitness\_array = fitness(population, n)

print("Fitness Array=>", fitness\_array )

#population = select(population, fitness\_array)

#print(population)

#print(fitness(population, n))

'''calling the GA function'''

print(GA(population, n, mutation\_threshold))