import numpy as np

import random

random.seed(2)

np.random.seed(8)

def fitness(population, n):

    fit\_list = []

    for individual in population:

        fit\_list.append(fit\_fun(individual))

    return fit\_list

def fit\_func(individual):

   n = len(individual)

    max\_goal\_state = (n\* (n- 1)) / 2

    attack = 0

    chess = np.zeros((n, n), dtype = int)

    for col in range(n):

        row = individual[col]

        chess[row][col] = 1

    for col in range(len(chess)):

        for row in range(len(chess)):

            if chess[row][col] == 1:

                for col\_right in range(col+ 1, len(chess)):

                    if chess[row][col\_right] == 1:

                        attack+= 1

                col\_right = col + 1

                row\_up = row - 1

                while col\_right < len(chess) and row\_up >= 0:

                    if chess[row\_up][column\_right] == 1:

                        attack+= 1

                    col\_right += 1

                    row\_up -= 1

                col\_right = col+ 1

                row\_down = row + 1

                while col\_right < len(chess) and row\_up < len(chess):

                    if chess[row\_up][col\_right] == 1:

                        attack += 1

                    col\_right += 1

                    row\_down += 1

    return max\_goal\_state - attacking\_pairs

def select(population, fit):

    a = list(range(0, len(population)))

    size = 1

    p = []

    total\_fit = sum(fit)

    for i in fit:

        p.append(i / total\_fit)

    index = np.random.choice(a, size, True, p)[0]

    return population[index]

def crossover(x, y):

    ran\_index = random.randint(0, len(x) - 1)

    child = []

    for i in range(0, ran\_index):

        child.append(x[i])

    for i in range(ran\_index, len(x)):

        child.append(y[i])

    return child

def mutate(child):

    ran\_index = random.ranint(0, len(child) - 1)

    ran\_val = random.ranint(0, len(child) - 1)

    new\_child = []

    for i in child:

        new\_child.append(i)

    new\_child[ran\_index] = ran\_val

    return new\_child

def GA(population, n, mutation\_threshold = 0.3):

    time = 0

    end\_time = 1000

    selected\_individual = None

    while True:

        new\_population = []

        fit\_list = fitness(population, len(population))

        m = len(population[0])

        goal = (m\* (m - 1)) / 2

        best\_fit = max(fit\_list)

        best\_index = -1

        for i in range(len(fit\_list)):

            if best\_fit == fit\_list[i]:

                best\_index = i

        if time >= end\_time or best\_fit== goal:

            selected\_individual = (time, best\_fit, population[best\_index])

            break

        for i in range(len(population)):

            x = select(population, fit\_list)

            y = select(population, fit\_list)

            child = crossover(x, y)

            ran\_num = random.random() # [0, 1]

            if (ran\_num < mutation\_threshold):

                child = mutate(child)

            new\_population.append(child)

        population = new\_population

        time += 1

    return selected\_individual

'''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

mutation\_threshold = 0.3

for n = 8 queen problem'''

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

solution = GA(population, n, mutation\_threshold)

print('Generation:', solution[0])

print('Fitness:', solution[1])

print('Arrangement:', solution[2])

```