import random

import numpy as np

def fitness\_function(populations, query):

    fitness\_list = []

    for temporary\_state in populations:

        # test 4

        # print("ekhane")

        max\_fitness = (query\*(query-1))/2

        x\_diagonal = []

        y\_diagonal = []

        # matrix for finding pairs easier

        matrix = np.zeros(shape=(len(temporary\_state), len(temporary\_state)))

        for i in range(len(temporary\_state)):

            x = temporary\_state[i]

            matrix[len(temporary\_state) - x - 1][i] = 1

            x\_diagonal.append(len(temporary\_state) - x)

            y\_diagonal.append(i)

        rows\_atk\_total = 0

        for i in range(len(temporary\_state)):

            # test 1

            # print("ami ekhane")

            sumRow = np.sum(matrix[i], dtype=int)

            attPairsInThisRow = (sumRow\*(sumRow-1))/2

            rows\_atk\_total += attPairsInThisRow

        diagonal\_atk\_total = 0

        for i in range(len(x\_diagonal)):

            # test 2

            # print("ami ebar ekhane")

            diagonal\_atk = 0

            sliced\_x = x\_diagonal[i + 1:]

            sliced\_y = y\_diagonal[i + 1:]

            for j in range(len(sliced\_x)):

                # test 3

                # print("ami ekebare ekkhane")

                if(abs(x\_diagonal[i]-sliced\_x[j]) == abs(y\_diagonal[i]-sliced\_y[j])):

                    diagonal\_atk = diagonal\_atk+1

            diagonal\_atk\_total = diagonal\_atk\_total+diagonal\_atk

        atk\_pairs = diagonal\_atk\_total + rows\_atk\_total

        fitness = max\_fitness - atk\_pairs

        fitness\_list.append(fitness)

    return fitness\_list

def crossover(x, y):

    # generating single random crossover point

    cross\_over\_point = np.random.randint(0, queens, dtype=int)

    child\_first = x[cross\_over\_point:]

    child\_second = y[:cross\_over\_point]

    return child\_first + child\_second

def mutation(channel):

    random\_index = np.random.randint(0, queens)

    random\_position = np.random.randint(0, queens)

    channel[random\_index] = random\_position

    return channel

def selection(pop, population\_fitness):

    fitness\_prob = []

    for i in range(len(population\_fitness)):

        fitness\_prob.append(

            (population\_fitness[i] / sum(population\_fitness)))

    a = [0 for i in range(len(population))]

    for i in range(len(population)):

        a[i] = i

    size = 1

    return pop[np.random.choice(a, size, True, fitness\_prob)[0]]

def state(query):

    pop = [random.randint(1, query) for i in range(query)]

    return pop

def genetic\_algo(pop, q, mut\_tr=0.3):

    max\_fitness = (q \* (q - 1))/2

    generation = 0

    while True:

        generation += 1

        new\_pop = []

        allFitness = fitness\_function(pop, q)

        if generation % 1000 == 0:

            print("Max fit -{} Generation {} ".format(max(allFitness), generation))

        if max(allFitness) == max\_fitness or generation == 200000:

            return (pop, allFitness, generation)

        for \_ in range(len(pop)):

            x = selection(pop, allFitness)

            y = selection(pop, allFitness)

            child = crossover(x, y)

            if(np.random.random() < mut\_tr):

                child = mutation(child)

            new\_pop.append(child)

        pop = new\_pop

if \_\_name\_\_ == "\_\_main\_\_":

    queens = int(input())

    starting\_population\_size = 10

    population = [state(queens) for i in range(starting\_population\_size)]

    popu, fit, generation = genetic\_algo(population, queens)

    print("Child {}, Max Fitness {}, Generation {}".format(

        popu[fit.index(max(fit))], max(fit), generation))