import random

import math

N = 5

POP\_SIZE = 10

GENERATIONS = 1000

MUTATION\_RATE = 0.1

class City:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def distance(city1, city2):

dx = city1.x - city2.x

dy = city1.y - city2.y

return math.sqrt(dx \* dx + dy \* dy)

def generate\_random\_permutation():

permutation = list(range(N))

random.shuffle(permutation)

return permutation

def calculate\_fitness(path, cities):

total\_distance = 0.0

for i in range(N - 1):

total\_distance += distance(cities[path[i]], cities[path[i + 1]])

total\_distance += distance(cities[path[N - 1]], cities[path[0]])

return total\_distance

def crossover(parent1, parent2):

pos1 = random.randint(0, N - 1)

pos2 = random.randint(0, N - 1)

if pos1 > pos2:

pos1, pos2 = pos2, pos1

child = [-1] \* N

child[pos1:pos2 + 1] = parent1[pos1:pos2 + 1]

j = 0

for i in range(N):

if j == pos1:

j = pos2 + 1

if parent2[i] not in child:

child[j] = parent2[i]

j += 1

return child

def mutate(path):

for i in range(N):

if random.random() < MUTATION\_RATE:

j = random.randint(0, N - 1)

path[i], path[j] = path[j], path[i]

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

random.seed()

cities = [City(0, 0), City(2, 2), City(2, 3), City(5, 5), City(4, 4)]

population = [generate\_random\_permutation() for \_ in range(POP\_SIZE)]

for generation in range(GENERATIONS):

fitness\_index = [(calculate\_fitness(individual, cities), i) for i, individual in enumerate(population)]

fitness\_index.sort()

new\_population = []

for \_ in range(POP\_SIZE):

parent1 = fitness\_index[random.randint(0, POP\_SIZE // 2 - 1)][1]

parent2 = fitness\_index[random.randint(0, POP\_SIZE // 2 - 1)][1]

child = crossover(population[parent1], population[parent2])

mutate(child)

new\_population.append(child)

population = new\_population

min\_distance = calculate\_fitness(population[0], cities)

best\_path = population[0]

for individual in population[1:]:

dist = calculate\_fitness(individual, cities)

if dist < min\_distance:

min\_distance = dist

best\_path = individual

print("Shortest path distance:", min\_distance)

print("Best path:", best\_path)