import random

class GeneticAlgorithm:

def \_\_init\_\_(self, file\_path, iterations):

self.file\_path = file\_path

self.target = None

self.batsman, self.avg\_run = self.populate\_batsman\_and\_avg\_run()

self.population = self.create\_initial\_population()

self.iterations = iterations

self.run\_genetic\_algorithm()

def populate\_batsman\_and\_avg\_run(self):

file = open(self.file\_path, "r")

first\_line = file.readline().split()

num\_of\_batsman, self.target = int(first\_line[0]), int(first\_line[1])

self.batsman, self.avg\_run = [0] \* num\_of\_batsman, [0] \* num\_of\_batsman

i = 0

for line in file:

line\_arr = line.split()

self.batsman[i], self.avg\_run[i] = line\_arr[0], int(line\_arr[1])

i += 1

return self.batsman, self.avg\_run

def create\_initial\_population(self):

population = []

for i in range(len(self.avg\_run)):

chromosome = []

for j in range(len(self.avg\_run)):

select = random.randint(0, 1)

chromosome.append(select)

population.append(chromosome)

return population

def run\_genetic\_algorithm(self):

iterations = 0

best\_chromosome = None

while iterations != self.iterations:

fitness = self.calculate\_fitness()

for i in range(len(fitness)):

if fitness[i] == 0:

best\_chromosome = self.population[i]

if best\_chromosome != None:

break

self.discard\_least\_fit(fitness)

random\_chromosome = self.selection()

self.crossover(random\_chromosome)

self.mutate()

iterations += 1

if best\_chromosome != None:

print(self.batsman)

print(best\_chromosome)

else:

print("-1")

def calculate\_fitness(self):

fitness = []

for chromosome in self.population:

total\_avg\_run = 0

for i in range(len(chromosome)):

if chromosome[i] == 1:

total\_avg\_run += self.avg\_run[i]

fitness.append(self.target - total\_avg\_run)

return fitness

def discard\_least\_fit(self, fitness):

max\_index = 0

for i in range(1, len(fitness)):

if fitness[i] > fitness[max\_index]:

max\_index = i

del self.population[max\_index]

def selection(self):

random\_index = random.randint(0, len(self.population) - 1)

return self.population.pop(random\_index)

def crossover(self, random\_chromosome):

new\_population = []

for chromosome in self.population:

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

first\_offspring = random\_chromosome[ : random\_index + 1] + chromosome[random\_index + 1 : ]

second\_offspring = chromosome[ : random\_index + 1] + random\_chromosome[random\_index + 1 : ]

new\_population.append(first\_offspring)

new\_population.append(second\_offspring)

self.population = new\_population

def mutate(self):

for chromosome in self.population:

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

mutation = random.randint(0, 1)

chromosome[random\_index] = mutation

GA = GeneticAlgorithm("Lab - 2/Input3.txt", 100)