import random

class DeliveryRouteOptimizer:

    def \_\_init\_\_(self, locations, distances, time\_windows, vehicle\_capacity, max\_iterations):

        self.locations = locations

        self.distances = distances

        self.time\_windows = time\_windows

        self.vehicle\_capacity = vehicle\_capacity

        self.max\_iterations = max\_iterations

        self.best\_solution = None

        self.best\_cost = float('inf')

    def calculate\_cost(self, solution):

        total\_distance = 0

        for i in range(len(solution) - 1):

            total\_distance += self.distances[self.locations[solution[i]]][self.locations[solution[i + 1]]]

        return total\_distance

    def generate\_initial\_solution(self):

        return random.sample(range(len(self.locations)), len(self.locations))

    def generate\_neighbour\_solution(self, current\_solution):

        new\_solution = current\_solution.copy()

        idx1, idx2 = random.sample(range(len(new\_solution)), 2)

        new\_solution[idx1], new\_solution[idx2] = new\_solution[idx2], new\_solution[idx1]

        return new\_solution

    def las\_vegas(self):

        for \_ in range(self.max\_iterations):

            current\_solution = self.generate\_initial\_solution()

            current\_cost = self.calculate\_cost(current\_solution)

            improved = True

            while improved:

                improved = False

                neighbour\_solution = self.generate\_neighbour\_solution(current\_solution)

                neighbour\_cost = self.calculate\_cost(neighbour\_solution)

                if neighbour\_cost < current\_cost:

                    current\_solution = neighbour\_solution

                    current\_cost = neighbour\_cost

                    improved = True

            if current\_cost < self.best\_cost:

                self.best\_solution = current\_solution

                self.best\_cost = current\_cost

    def get\_best\_solution(self):

        return self.best\_solution, self.best\_cost

# User input

num\_locations = int(input("Enter the number of locations: "))

locations = [input(f"Enter location {i+1}: ") for i in range(num\_locations)]

distances = {}

for loc in locations:

    distances[loc] = {}

    for dest in locations:

        if loc != dest:

            distances[loc][dest] = int(input(f"Enter distance from {loc} to {dest}: "))

time\_windows = {}

for loc in locations:

    start, end = map(int, input(f"Enter time window for {loc} (start end): ").split())

    time\_windows[loc] = (start, end)

vehicle\_capacity = int(input("Enter vehicle capacity: "))

max\_iterations = int(input("Enter maximum iterations: "))

# Route optimization

optimizer = DeliveryRouteOptimizer(locations, distances, time\_windows, vehicle\_capacity, max\_iterations)

optimizer.las\_vegas()

best\_solution, best\_cost = optimizer.get\_best\_solution()

# Output

print("Best Solution:", [locations[i] for i in best\_solution])

print("Best Cost:", best\_cost)

from tabulate import tabulate

# Output

table\_data = [[f"Location {i+1}", locations[i]] for i in range(len(locations))]

print("\nLocations:")

print(tabulate(table\_data, headers=["Index", "Location"], tablefmt="fancy\_grid"))

print("\nBest Solution:")

print(tabulate([[f"Location {i+1}", locations[idx]] for i, idx in enumerate(best\_solution)],

               headers=["Index", "Location"], tablefmt="fancy\_grid"))

print("\nBest Cost:", best\_cost)