```python
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""
@Project : Study
@File :    遗传代码实现.py
@Author :  少年又远方
@Date :    2022/4/28 17:17
@LastDate:
@Description:
"""
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import time

start = time.time()

# 31个城市的坐标
city_condition = [[106.54, 29.59], [91.11, 29.97], [87.68, 43.77], [106.27, 38.47], [111.65, 40.82],
                  [108.33, 22.84], [126.63, 45.75], [125.35, 43.88], [123.38, 41.8], [114.48, 38.03], [112.53, 37.87],
                  [101.74, 36.56], [117.0, 36.65], [113.6, 34.76], [118.78, 32.04], [117.27, 31.86],
                  [120.19, 30.26], [119.3, 26.08], [115.89, 28.68], [113.0, 28.21], [114.31, 30.52],
                  [113.23, 23.16], [121.5, 25.05], [110.35, 20.02], [103.73, 36.03], [108.95, 34.27],
                  [104.06, 30.67], [106.71, 26.57], [102.73, 25.04], [114.1, 22.2], [113.33, 22.13]]

# 距离矩阵
# 使用numpy计算生成距离矩阵:
city_count = 31
Distance = np.zeros([city_count, city_count])
for i in range(city_count):
    for j in range(city_count):
        Distance[i][j] = math.sqrt(
            (city_condition[i][0] - city_condition[j][0]) ** 2 + (city_condition[i][1] - city_condition[j][1]) ** 2)
# 种群数
count = 200
# 改良次数
improve_count = 500
# 进化次数
iteration = 2000
# 设置强者的定义概率，即种群前20%为强者
retain_rate = 0.2
# 变异率
mutation_rate = 0.1
# 设置起点
index = [i for i in range(city_count)]


# 总距离
def get_total_distance(path_new):
    distance = 0
    for i in range(city_count - 1):
        # count为30，意味着回到了开始的点，此时的值应该为0.
        distance += Distance[int(path_new[i])][int(path_new[i + 1])]
    distance += Distance[int(path_new[-1])][int(path_new[0])]
    return distance


# 改良
# 思想：随机生成两个城市，任意交换两个城市的位置，如果总距离减少，就改变染色体。
# 此处不必关心，此函数用于种群的初始化
def improve(x):
    i = 0
    distance = get_total_distance(x)
    while i < improve_count:
        u = random.randint(0, len(x) - 1)
        v = random.randint(0, len(x) - 1)
        if u != v:
            new_x = x.copy()
            # # 随机交叉两个点，t为中间数
            t = new_x[u]
            new_x[u] = new_x[v]
            new_x[v] = t
            new_distance = get_total_distance(new_x)
            if new_distance < distance:
                distance = new_distance
                x = new_x.copy()
        else:
            continue
        i += 1


# 适应度评估，选择，迭代一次选择一次
def selection(population):
    # 对总距离从小到大进行排序
    graded = [[get_total_distance(x), x] for x in population]
    graded = [x[1] for x in sorted(graded, key=lambda x: x[0])]
    '''for item in graded:
        item0 = sorted(item)
        check = list(range(31))
        print(item0==check)
    '''
    # 选出适应性强的染色体
    # 注记:此处的x为列表，列表的第一个元素为总距离，第二个元素为染色体
    # ********** Begin **********#
    retain_length = math.floor(len(graded)*retain_rate)
    # 适应度强的集合,直接加入选择中
    parents = graded[:retain_length+1]
    # 轮盘赌算法选出K个适应性不强的个体，保证种群的多样性
    s = graded[retain_length:]
    # 挑选的不强的个数
    k = count * 0.2
```

```
    # 存储适应度
    a = []
    for i in range(0, len(s)):
        a.append(get_total_distance(s[i]))
    sum = np.sum(a)
    b = np.cumsum(a / sum)
    while k > 0:  # 迭代一次选择k条染色体
        t = random.random()
        for h in range(1, len(b)):
            if b[h - 1] < t <= b[h]:
                parents.append(s[h])
                k -= 1
                break
    return parents


# 交叉繁殖
def crossover(parents):
    # 生成子代的个数,以此保证种群稳定
    target_count = count - len(parents)
    # 孩子列表
    children = []
    while len(children) < target_count:
        male_index = random.randint(0, len(parents) - 1)
        female_index = random.randint(0, len(parents) - 1)
        # 在适应度强的中间选择父母染色体
        if male_index != female_index:
            male = parents[male_index]
            female = parents[female_index]
            left = random.randint(0, len(male) - 2)
            right = random.randint(left + 1, len(male) - 1)
            # print(female)
            # 交叉片段
            gene1 = male[left:right]
            gene2 = female[left:right]
            # 得到原序列通过改变序列的染色体，并复制出来备用。
            child1_c = male[right:] + male[:right]     # P3
            child2_c = female[right:] + female[:right] # P4
            child1 = child1_c.copy()
            child2 = child2_c.copy()

            # 已经改变的序列=>去掉交叉片段后的序列
            for o in gene2:
                child1_c.remove(o)
            for o in gene1:
                child2_c.remove(o)
            # 交换交叉片段
            seg = len(male)-1-right
            new_child1 = child1_c[seg:]+gene2+child1_c[:seg]
            new_child2 = child2_c[seg:]+gene1+child2_c[:seg]
            children.append(new_child1)
            children.append(new_child2)
    return children


# 变异
def mutation(children):
    # children现在包括交叉和优质的染色体
    for i in range(len(children)):
        if random.random() < mutation_rate:
            child = children[i]
            # 产生随机数
            u = random.randint(0, len(child) - 4)
            v = random.randint(u + 1, len(child) - 3)
            w = random.randint(v + 1, len(child) - 2)
            # 采用切片操作实现变异，将下标在u，v之间的数值插入下标为w，元素后面
            child = child[0:u] + child[v:w] + child[u:v] + child[w:]
            children[i] = child
    return children


# 得到最佳纯输出结果
def get_result(population):
    graded = [[get_total_distance(x), x] for x in population]
    graded = sorted(graded)
    return graded[0][0], graded[0][1]


if __name__ == '__main__':
    # 使用改良圈算法初始化种群
    population = []
    for i in range(count):
        # 随机生成个体
        x = index.copy()
        # 随机排序
        random.shuffle(x)
        improve(x)
        population.append(x)
    # 主函数:
    register = []
    i = 0
    distance, result_path = get_result(population)
    register.append(distance)
    while i < iteration:
        # 选择繁殖个体群
        parents = selection(population)
        # 交叉繁殖
        children = crossover(parents)
        # 变异操作
        children = mutation(children)
        # 更新种群
        population = parents + children
```

```
        distance, result_path = get_result(population)
        register.append(distance)
        if i%50 == 0:
            print("第{}次迭代：".format(i))
            print("当前距离：", distance)
            print("当前路径：", result_path)
        i = i + 1
    print("迭代次数：", iteration)
    print("最优值是：", distance)
    print("最优路径：", result_path)
    if distance < 170:
        print("路径长度符合要求")
    else:
        print("路径长度太长")
    plt.rcParams['font.sans-serif'] = 'SimHei'  # 设置中文显示
    plt.rcParams['axes.unicode_minus'] = False
    plt.figure(1)
    X = []
    Y = []
    for item in result_path:
        X.append(city_condition[item][0])
        Y.append(city_condition[item][1])
    plt.plot(X, Y, '-o')
    for i in range(len(X)):
        plt.text(X[i] + 0.05, Y[i] + 0.05, str(result_path[i]), color='red')
    plt.xlabel('横坐标')
    plt.ylabel('纵坐标')
    plt.title('轨迹图')
    plt.show()
    plt.figure(2)
    plt.plot(np.array(register))
    plt.title('优化过程')
    plt.ylabel('最优值')
    plt.xlabel('代数({}->{})'.format(0,  iteration))
```