#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define N 5 // Number of cities

#define POP\_SIZE 10

#define GENERATIONS 1000

#define MUTATION\_RATE 0.1

// Structure to represent a city

struct City {

int x, y;

};

// Function to calculate the Euclidean distance between two cities

double distance(struct City city1, struct City city2) {

int dx = city1.x - city2.x;

int dy = city1.y - city2.y;

return sqrt(dx \* dx + dy \* dy);

}

// Function to generate random permutation of cities

void generateRandomPermutation(int permutation[N]) {

for (int i = 0; i < N; ++i)

permutation[i] = i;

for (int i = N - 1; i > 0; --i) {

int j = rand() % (i + 1);

int temp = permutation[i];

permutation[i] = permutation[j];

permutation[j] = temp;

}

}

// Function to calculate the fitness (total distance) of an individual (path)

double calculateFitness(int path[N], struct City cities[N]) {

double totalDistance = 0.0;

for (int i = 0; i < N - 1; ++i)

totalDistance += distance(cities[path[i]], cities[path[i + 1]]);

totalDistance += distance(cities[path[N - 1]], cities[path[0]]);

return totalDistance;

}

// Function to perform crossover (order crossover)

void crossover(int parent1[N], int parent2[N], int child[N]) {

int pos1 = rand() % N;

int pos2 = rand() % N;

if (pos1 > pos2) {

int temp = pos1;

pos1 = pos2;

pos2 = temp;

}

for (int i = pos1; i <= pos2; ++i)

child[i] = parent1[i];

int j = 0;

for (int i = 0; i < N; ++i) {

if (j == pos1) j = pos2 + 1;

int found = 0;

for (int k = 0; k < N; ++k) {

if (child[k] == parent2[i]) {

found = 1;

break;

}

}

if (!found) {

child[j++] = parent2[i];

}

}

}

// Function to perform mutation (swap mutation)

void mutate(int path[N]) {

for (int i = 0; i < N; ++i) {

if ((rand() / (double)RAND\_MAX) < MUTATION\_RATE) {

int j = rand() % N;

int temp = path[i];

path[i] = path[j];

path[j] = temp;

}

}

}

int main() {

srand(time(NULL));

// Define the cities

struct City cities[N] = {{0, 5}, {1, 5}, {2, 3}, {5, 1}, {3, 3}};

// Initialize population

int population[POP\_SIZE][N];

for (int i = 0; i < POP\_SIZE; ++i)

generateRandomPermutation(population[i]);

for (int generation = 0; generation < GENERATIONS; ++generation) {

// Calculate fitness for each individual

double fitnessIndex[POP\_SIZE];

for (int i = 0; i < POP\_SIZE; ++i)

fitnessIndex[i] = calculateFitness(population[i], cities);

// Create new population through selection, crossover, and mutation

int newPopulation[POP\_SIZE][N];

for (int i = 0; i < POP\_SIZE; ++i) {

int parent1 = rand() % (POP\_SIZE / 2);

int parent2 = rand() % (POP\_SIZE / 2);

crossover(population[parent1], population[parent2], newPopulation[i]);

mutate(newPopulation[i]);

}

for (int i = 0; i < POP\_SIZE; ++i)

for (int j = 0; j < N; ++j)

population[i][j] = newPopulation[i][j];

}

// Print the best solution (shortest path)

double minDistance = calculateFitness(population[0], cities);

int bestPath[N];

for (int i = 0; i < N; ++i)

bestPath[i] = population[0][i];

for (int i = 1; i < POP\_SIZE; ++i) {

double dist = calculateFitness(population[i], cities);

if (dist < minDistance) {

minDistance = dist;

for (int j = 0; j < N; ++j)

bestPath[j] = population[i][j];

}

}

printf("Shortest path distance: %f\n", minDistance);

printf("Best path: ");

for (int i = 0; i < N; ++i)

printf("%d ", bestPath[i]);

printf("\n");

return 0;

}