// This C++ code is created by Ali Jawad Ibada.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <stdio.h>
#include <algorithm>
#include <numeric>
#include<time.h>

using namespace std;

int p = 4;                                  // individual number for printr && sum
int Radius = 100;                           // Radius of C_group
const int N_ind = 5;                        // number of individuals in population
const int N_clones = 10;                    // number of clones
const int I_segment = 5;                    // segment size
int  nn_ls = 40;

int N_inf = 10;                             // number of infection in gene transfer
const int I_transfer = 50;                  // length of segment in gene transfer

void rvereseArray(int arr[], int start, int end)
{
        int temp;
        while (start < end)
        {
                temp = arr[start];
                arr[start] = arr[end];
                arr[end] = temp;
                start++;
                end--;
        }
}

class DBMEA
{
        int I_seg[I_transfer];                  //copy of good segment in gene transfer
        double Csum_matrix[N_ind];              //Cost sum matrix
        int a3[I_segment] = { 0 };              //copy of I_segment for loose mutation

        double* C_sum;                          //clons cost
        int ncities;                            //number of cities
        int* a;                                 //random route for population and loose mutation
        int* tour;                              //temp route for local search
        int* tour1;                             //temp route for local search
        int* tour2;                             //temp route for local search
        int* pos;

        int** Population;                   //Route matrix (population)
        int** Population1;                  //copy of Route matrix (population)
        double** c;                         //cost matrix
        double** cc;                        //copy of cost matrix
        int** clone;                        //clones matrix
        int** nnMat;                        //nearest neighbor matrix

public:
        DBMEA(int ncities)
        {
                this->ncities = ncities;
                c = new double* [ncities];
                cc = new double* [ncities];
```

```
	for (int i = 0; i < ncities; i++)
	{
		c[i] = new double[ncities];
		memset(c[i], false, ncities * sizeof(int));
		cc[i] = new double[ncities];
		memset(cc[i], false, ncities * sizeof(int));
	}

	Population = new int* [N_ind];
	for (int i = 0; i < N_ind; i++)
	{
		Population[i] = new int[ncities];
		memset(Population[i], false, ncities * sizeof(int));
	}

	Population1 = new int* [N_ind];
	for (int i = 0; i < N_ind; i++)
	{
		Population1[i] = new int[ncities];
		memset(Population1[i], false, ncities * sizeof(int));
	}

	clone = new int* [N_clones];
	for (int i = 0; i < N_clones; i++)
	{
		clone[i] = new int[ncities];
		memset(clone[i], false, ncities * sizeof(int));
	}

	nnMat = new int* [ncities];
	for (int i = 0; i < ncities; i++)
	{
		nnMat[i] = new int[ncities];
		memset(nnMat[i], false, ncities * sizeof(int));
	}

	a = new int[ncities - 1];
	C_sum = new double[N_clones];

	tour = new int[ncities + 1];
	tour1 = new int[ncities + 1];
	tour2 = new int[ncities + 1];
	pos = new int[ncities];
}

void Create_cost(vector<double>x, vector<double>y)
{
	for (int i = 0; i < ncities; i++)
	{
		for (int j = i + 1; j < ncities; j++)
		{
			c[i][j] = sqrt(pow(x[j] - x[i], 2) + pow(y[j] - y[i], 2));
			c[j][i] = c[i][j];
		}
		c[i][i] = 10e10;
	}
	int a, b;
	double y1, z;
	for (int i = 0; i < ncities; i++)
	{
		for (int j = 0; j < ncities; j++)
		{
			cc[i][j] = c[i][j];
```

```
                        nnMat[i][j] = j;
                }
                nnMat[i][0] = i;
        }


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

                for (int i = ncities; i > 1; i--) {
                        for (int j = 1; j < i - 1; j++) {

                                if (cc[k][j] > cc[k][j + 1])
                                {
                                        z = cc[k][j];
                                        y1 = cc[k][j + 1];
                                        a = nnMat[k][j];
                                        b = nnMat[k][j + 1];
                                        cc[k][j] = y1;
                                        cc[k][j + 1] = z;
                                        nnMat[k][j] = b;
                                        nnMat[k][j + 1] = a;

                                }
                        }
                }
        }
}

void NN()
{
        int m;
        for (int i = 0; i < ncities; i++)
        {
                for (int j = 0; j < ncities; j++)
                {
                        cc[i][j] = c[i][j];
                }
        }

        Population[0][0] = 0;
        for (int i = 0; i < ncities - 1; i++)
        {
                m = min_element(cc[Population[0][i]], cc[Population[0][i]] + ncities) - cc[Population[0][i]];
                cc[Population[0][i]][m] = 10e10;
                Population[0][i + 1] = m;

                for (int j = 0; j < i + 1; j++)
                {
                        if (Population[0][j] == m)
                        {
                                i--;
                                break;
                        }
                }
        }
        cout << "NN = " << Csum(Population[0]) << endl;
}

void SNN()
{
        int m, a1;
        for (int i = 0; i < ncities; i++)
        {
```

```
		for (int j = 0; j < ncities; j++)
		{
			cc[i][j] = c[i][j];
		}
	}

	Population[1][0] = 0;

	for (int i = 0; i < ncities - 1; i++)
	{
		a1 = 1;
		while (a1 == 1 && i < ncities - 2)
		{
			a1 = 0;
			m = min_element(cc[Population[1][i]], cc[Population[1][i]] + ncities) - cc[Population[1][i]];
			cc[Population[1][i]][m] = 10e10;
			for (int j = 0; j < i + 1; j++)
			{
				if (Population[1][j] == m)
				{
					a1 = 1;
					break;
				}
			}
		}
		do
		{
			a1 = 0;
			m = min_element(cc[Population[1][i]], cc[Population[1][i]] + ncities) - cc[Population[1][i]];
			cc[Population[1][i]][m] = 10e10;
			for (int j = 0; j < i + 1; j++)
			{
				if (Population[1][j] == m)
				{
					a1 = 1;
					break;
				}
			}
		} while (a1 == 1);
		Population[1][i + 1] = m;
	}
	cout << "SNN = " << Csum(Population[1]) << endl;
}

void ANN()
{
	int m, a1;
	for (int i = 0; i < ncities; i++)
	{
		for (int j = 0; j < ncities; j++)
		{
			cc[i][j] = c[i][j];
		}
	}

	Population[2][0] = 0;
	for (int i = 0; i < ncities - 1; i++)
	{
		if (i % 2 == 0)
		{
			do
			{
				a1 = 0;
```

```
                                        m = min_element(cc[Population[2][i]], cc[Population[2][i]] + ncities) -
cc[Population[2][i]];
                                        cc[Population[2][i]][m] = 10e10;
                                        for (int j = 0; j < i + 1; j++)
                                        {
                                                if (Population[2][j] == m)
                                                {
                                                        a1 = 1;
                                                        break;
                                                }
                                        }
                                } while (a1 == 1);
                                Population[2][i + 1] = m;
                        }
                        else
                        {
                                a1 = 1;
                                while (a1 == 1 && i < ncities - 2)
                                {
                                        a1 = 0;
                                        m = min_element(cc[Population[2][i]], cc[Population[2][i]] + ncities) -
cc[Population[2][i]];
                                        cc[Population[2][i]][m] = 10e10;
                                        for (int j = 0; j < i + 1; j++)
                                        {
                                                if (Population[2][j] == m)
                                                {
                                                        a1 = 1;
                                                        break;
                                                }
                                        }
                                }
                                do
                                {
                                        a1 = 0;
                                        m = min_element(cc[Population[2][i]], cc[Population[2][i]] + ncities) -
cc[Population[2][i]];
                                        cc[Population[2][i]][m] = 10e10;
                                        for (int j = 0; j < i + 1; j++)
                                        {
                                                if (Population[2][j] == m)
                                                {
                                                        a1 = 1;
                                                        break;
                                                }
                                        }
                                } while (a1 == 1);
                                Population[2][i + 1] = m;
                        }
                }
                cout << "ANN1 = " << Csum(Population[2]) << endl;
        }

        void C_group()
        {
                for (int i = 0; i < ncities; i++)
                {
                        for (int j = 0; j < ncities; j++)
                        {
                                cc[i][j] = c[i][j];
                        }
                }
```

```
bool a1, a2;
int last = 0, count = 0, m, aa[500] = {}, index, par1;
double temp1, temp2;
Population[4][0] = 0;
for (int i = 0; i < ncities - 1; i++)
{
	Population[4][i + 1] = 0;
	do
	{
		a1 = 0;
		m = min_element(cc[last], cc[last] + ncities) - cc[last];
		temp1 = cc[last][m];
		cc[last][m] = 10e10;
		for (int j = 0; j <= i; j++)
		{
			if (Population[4][j] == m)
			{
				a1 = 1;
				break;
			}
		}
	} while (a1 == 1);

	if (temp1 < Radius)
	{
		aa[count] = m;
		count++;
		a2 = 0;
		Population[4][i + 1] = m;
	}
	else
	{
		a2 = 1;
	}

	if ((count != 0) && (a2 == 1 || i == ncities - 2))
	{
		if (i == ncities - 2 && a2 == 0)
			i++;
		Population[4][i - count + 1] = aa[0];
		par1 = aa[0];
		for (int j = 0; j < count - 1; j++)
		{
			temp2 = 10e10;
			for (int k = 1; k < count; k++)
			{
				if ((temp2 > c[par1][aa[k]]) && (aa[k] != 0))
				{
					temp2 = c[par1][aa[k]];
					index = k;
				}
			}
			par1 = aa[index];
			Population[4][i - count + 2 + j] = aa[index];
			aa[index] = 0;

		}
		count = 0;
		Population[4][i + 1] = m;
		last = m;
	}
	else if (a2 == 1)
	{
```

```
                        Population[4][i + 1] = m;
                        last = m;
                }
        }
        cout << "C_group = " << Csum(Population[4]) << endl;
}

void population()
{
        srand(time(0));
        iota(&a[0], &a[ncities - 1], 1);
        random_shuffle(&a[0], &a[ncities - 1]);

        for (int i = 5; i < N_ind; i++)
        {
                random_shuffle(&a[0], &a[ncities - 1]);
                copy(&a[0], &a[ncities - 1], &Population[i][1]);
        }
        cout << endl << "Total cost is = " << Csum(Population[p]) << endl;
}

void mutation_coherent()
{
        int i, m1;   // m1 : the cost index of the clone
        int m2;      // m2 :index of the last segment
        for (int i1 = 0; i1 < N_ind; i1++)
        {
                for (i = 0; i < N_clones; i++)
                        copy(&Population[i1][0], &Population[i1][ncities], &clone[i][0]);

                for (i = 1; i < ncities - I_segment; i += I_segment)
                {
                        C_sum[0] = Csum(clone[0]);
                        for (int j = 1; j <= I_segment; j++)  //reverse 1
                                clone[1][j + i - 1] = clone[0][I_segment - j + i];
                        C_sum[1] = Csum(clone[1]);
                        for (int j = 2; j < N_clones; j++)
                        {
                                random_shuffle(&clone[j][i], &clone[j][I_segment + i]);
                                C_sum[j] = Csum(clone[j]);
                        }
                        m1 = min_element(&C_sum[0], &C_sum[N_clones]) - &C_sum[0];
                        if (m1 != 0)
                                copy(&clone[m1][i], &clone[m1][I_segment + i], &clone[0][i]);

                        for (int j = 1; j < N_clones; j++)
                                copy(&clone[0][i], &clone[0][I_segment + i], &clone[j][i]);
                }

                C_sum[0] = Csum(clone[0]);
                m2 = 1;
                for (int j = i; j < ncities; j++)  //reverse 1
                {
                        clone[1][j] = clone[0][ncities - m2];
                        m2++;
                }
                C_sum[1] = Csum(clone[1]);

                for (int j = 2; j < N_clones; j++)
                {
                        random_shuffle(&clone[j][i], &clone[j][ncities]);
                        C_sum[j] = Csum(clone[j]);
                }
```

```
                m1 = min_element(&C_sum[0], &C_sum[N_clones]) - &C_sum[0];
                if (m1 != 0)
                        copy(&clone[m1][i], &clone[m1][ncities], &clone[0][i]);

                for (int j = 1; j < N_clones; j++)
                        copy(&clone[0][i], &clone[0][ncities], &clone[j][i]);

                copy(&clone[0][0], &clone[0][ncities], &Population[i1][0]);      //copy clone 0 to r
        }
        cout << "Coherent Mutation = " << Csum(Population[p]) << endl;
}

void mutation_loose()
{
        int i, m1;   // m1 : the cost index of the clone
        int m2;     // m2 :index of the last segment

        for (int i1 = 0; i1 < N_ind; i1++)
        {
                for (i = 0; i < N_clones; i++)
                        copy(&Population[i1][0], &Population[i1][ncities], &clone[i][0]);

                for (i = 0; i < ncities - l_segment - 1; i += l_segment)
                {
                        C_sum[0] = Csum(clone[0]);
                        for (int j = 0; j < l_segment; j++)  //reverse 1
                                clone[1][a[j + i]] = clone[0][a[l_segment + i - j - 1]];
                        C_sum[1] = Csum(clone[1]);

                        for (int j = 2; j < N_clones; j++)
                        {
                                copy(&a[i], &a[l_segment + i], &a3[0]);
                                random_shuffle(&a[i], &a[l_segment + i]);

                                for (int x1 = 0; x1 < l_segment; x1++)
                                {
                                        clone[j][a[x1 + i]] = clone[0][a3[x1]];
                                }
                                C_sum[j] = Csum(clone[j]);
                        }
                        m1 = min_element(&C_sum[0], &C_sum[N_clones]) - &C_sum[0];
                        if (m1 != 0)
                        {
                                for (int x2 = 0; x2 < l_segment; x2++)
                                        clone[0][a[x2 + i]] = clone[m1][a[x2 + i]];
                        }

                        for (int x3 = 1; x3 < N_clones; x3++)
                                for (int x4 = 0; x4 < l_segment; x4++)
                                        clone[x3][a[i + x4]] = clone[0][a[i + x4]];
                }

                C_sum[0] = Csum(clone[0]);
                m2 = 2;
                for (int j = i; j < ncities - 1; j++)  //reverse 1
                {
                        clone[1][a[j]] = clone[0][a[ncities - m2]];
                        m2++;
                }
                C_sum[1] = Csum(clone[1]);
                for (int j = 2; j < N_clones; j++)
                {
```

```
                        copy(&a[i], &a[ncities - 1], &a3[0]);
                        random_shuffle(&a[i], &a[ncities - 1]);

                        for (int x1 = 0; x1 < ncities - i - 1; x1++)
                        {
                                clone[j][a[x1 + i]] = clone[0][a3[x1]];
                        }
                        C_sum[j] = Csum(clone[j]);
                }

                m1 = min_element(&C_sum[0], &C_sum[N_clones]) - &C_sum[0];
                if (m1 != 0)
                {
                        for (int x2 = 0; i + x2 < ncities - 1; x2++)
                                clone[0][a[x2 + i]] = clone[m1][a[x2 + i]];
                }

                for (int x3 = 1; x3 < N_clones; x3++)
                        for (int x4 = 1; i + x4 < ncities; x4++)
                                clone[x3][a[i + x4 - 1]] = clone[0][a[i + x4 - 1]];

                copy(&clone[0][0], &clone[0][ncities], &Population[i1][0]);      //copy clone 0 to r
        }
        cout << "Loose Mutation = " << Csum(Population[p]) << endl;
}

void local_search()
{
        int r[50];
        double length;
        int i;
        int j;
        int h;
        int l;
        int m;
        for (int i = 0; i < 5; i++) {
                r[i] = i;// rand() % (N_ind - N_ind / 2) + (N_ind / 2);
        }
        Csum_all();
        for (m = 0; m < 5; m++) {

                for (j = 0; j < ncities - 1; j++) {
                        tour[0] = 0;
                        tour[j + 1] = Population[r[m]][j + 1];
                        tour[ncities] = 0;
                }
                length = Csum_matrix[r[m]];

                for (i = 0; i < ncities; i++) {
                        pos[tour[i]] = i;
                }
                // 2-opt
                int c1, c2;        // c1, c2 az élcseréhez
                int suc_c1, suc_c2;     // c1 c2 successora
                int pred_c1, pred_c2;     //c1 c2 predecessora
                int pos_c1, pos_c2;     // c1, c2 pozíciója
                int improvement_flag, help, n_exchanges = 0;
                int h1 = 0, h2 = 0, h3 = 0, h4 = 0;
                double radius;            // keresés sugara
                double gain = 0;
                double max_gain = 0;
                double x = 0, y = 0, z = 0, zs = 0;
                improvement_flag = 1;
```

```
int n_improves = 0;

while (improvement_flag) {
	improvement_flag = 0;
	max_gain = 0;
	for (l = 0; l < ncities; l++) {
		c1 = l;
		pos_c1 = pos[c1];
		suc_c1 = tour[pos_c1 + 1];
		radius = c[c1][suc_c1];
		for (h = 0; h < nn_ls; h++) {

			c2 = nnMat[c1][h];
			if (radius > c[c1][c2]) {
				suc_c2 = tour[pos[c2] + 1];
				gain = -radius + c[c1][c2] +
					c[suc_c1][suc_c2] - c[c2][suc_c2];
				if (gain > -10e-10) {
					gain = 0;
				}
				if (gain < max_gain) {
					h1 = c1; h2 = suc_c1; h3 = c2; h4 = suc_c2;
					improvement_flag = 1;
					max_gain = gain;
				}
			}
			else
				break;
		}

		if (pos_c1 > 0)
			pred_c1 = tour[pos_c1 - 1];
		else
			pred_c1 = tour[ncities - 1];
		radius = c[pred_c1][c1];
		for (h = 0; h < nn_ls; h++) {
			c2 = nnMat[c1][h];
			if (radius > c[c1][c2]) {
				pos_c2 = pos[c2];
				if (pos_c2 > 0)
					pred_c2 = tour[pos_c2 - 1];
				else
					pred_c2 = tour[ncities - 1];
				if (pred_c2 == c1)
					continue;
				if (pred_c1 == c2)
					continue;
				gain = -radius + c[c1][c2] +
					c[pred_c1][pred_c2] - c[pred_c2][c2];
				if (gain > -10e-10) {
					gain = 0;
				}
				if (gain < max_gain) {
					h1 = pred_c1; h2 = c1; h3 = pred_c2; h4 = c2;
					improvement_flag = 1;
					max_gain = gain;
				}
			}
			else
				break;
		}
	}
	if (improvement_flag) {
```

```
                        n_exchanges++;
                        length += max_gain;

                        if (pos[h3] < pos[h1]) {
                                help = h1; h1 = h3; h3 = help;
                                help = h2; h2 = h4; h4 = help;
                        }
                        if (pos[h3] - pos[h2] < ncities / 2 + 1) {

                                i = pos[h2]; j = pos[h3];
                                while (i < j) {
                                        c1 = tour[i];
                                        c2 = tour[j];
                                        tour[i] = c2;
                                        tour[j] = c1;
                                        pos[c1] = j;
                                        pos[c2] = i;
                                        i++; j--;
                                }
                        }
                        else {
                                i = pos[h1]; j = pos[h4];
                                if (j > i)
                                        help = ncities - (j - i) + 1;
                                else
                                        help = (i - j) + 1;
                                help = help / 2;
                                for (h = 0; h < help; h++) {
                                        c1 = tour[i];
                                        c2 = tour[j];
                                        tour[i] = c2;
                                        tour[j] = c1;
                                        pos[c1] = j;
                                        pos[c2] = i;
                                        i--; j++;
                                        if (i < 0)
                                                i = ncities - 1;
                                        if (j >= ncities)
                                                j = 0;
                                }
                                tour[ncities] = tour[0];
                        }
                }
        }
        length = 0;
        for (int i = 0; i < ncities; i++) {
                length = length + c[tour[i]][tour[i + 1]];
        }

        for (int i = 0; i < ncities - pos[0]; i++) {
                tour1[i] = tour[pos[0] + i];
        }
        for (int i = 0; i < pos[0]; i++) {
                tour1[ncities - pos[0] + i] = tour[i];
        }
        tour1[ncities] = 0;

        for (int i = 0; i < ncities + 1; i++) {
                tour[i] = tour1[i];
        }

        for (i = 0; i < ncities; i++) {
                pos[tour[i]] = i;
```

```
}

/* 3-opt */
double zmin = -1;
int a, b, c_1, c_2, k;
int A1, B1, C1, D1, E1, F1;
double z_1 = 0, z_2 = 0, z1, z2, z1_min, z2_min, z_diff, z_diff1, z_diff2;

if (m < 10) {
	while (zmin < 0.0) {
		zmin = 0.0;
		i = 1;
		l = ncities - 1;
		while (i <= ncities - 4) {
			a = i;
			i++;
			for (h = 0; h < nn_ls; h++) {
				b = pos[nnMat[tour[a - 1]][h]];
				for (k = 0; k < nn_ls; k++) {
					c_1 = pos[nnMat[tour[a]][k]] + 1;
					c_2 = pos[nnMat[tour[a]][k]];
					if (b >= a + 2 && c_1 >= b + 2)
					{
						z_1 = c[tour[a - 1]][tour[a]] + c[tour[b -
1]][tour[b]] + c[tour[c_1 - 1]][tour[c_1]];
						z1 = c[tour[a - 1]][tour[b]] + c[tour[c_1 -
1]][tour[a]] + c[tour[b - 1]][tour[c_1]];
					}
					else
					{
						z1 = 10e10;
					}
					if (b >= a + 2 && c_2 >= b + 2)
					{
						z_2 = c[tour[a - 1]][tour[a]] + c[tour[b -
1]][tour[b]] + c[tour[c_2 - 1]][tour[c_2]];
						z2 = c[tour[a - 1]][tour[b]] + c[tour[c_2 -
1]][tour[b - 1]] + c[tour[a]][tour[c_2]];
					}
					else
					{
						z2 = 10e10;
					}
					z_diff1 = z1 - z_1;
					z_diff2 = z2 - z_2;
					z_diff = min(z_diff1, z_diff2);

					if (z_diff < zmin) {
						zmin = z_diff;
						z1_min = z_diff1;
						z2_min = z_diff2;
						if (z1_min < z2_min)
						{
							A1 = a - 1;
							B1 = a;
							C1 = b - 1;
							D1 = b;
							E1 = c_1 - 1;
							F1 = c_1;

						}
```

```
                                                                else
                                                                {
                                                                        A1 = a - 1;
                                                                        B1 = a;
                                                                        C1 = b - 1;
                                                                        D1 = b;
                                                                        E1 = c_2 - 1;
                                                                        F1 = c_2;

                                                                }
                                                        }
                                                }
                                        }
                                }
                                if (zmin < 0.0) {
                                        if (z1_min < z2_min)
                                        {
                                                copy(tour + 0, tour + A1 + 1, tour2 + 0);
                                                copy(tour + D1, tour + E1 + 1, tour2 + A1 + 1);
                                                copy(tour + B1, tour + C1 + 1, tour2 + A1 + E1 - D1 + 2);
                                                copy(tour + F1, tour + ncities + 1, tour2 + A1 + E1 - D1 + C1 - B1 +
3);

                                                memcpy(&tour[0], &tour2[0], (ncities + 1) * sizeof(int));

                                        }
                                        else
                                        {
                                                copy(tour + 0, tour + A1 + 1, tour2 + 0);
                                                copy(tour + D1, tour + E1 + 1, tour2 + A1 + 1);
                                                copy(tour + B1, tour + C1 + 1, tour2 + A1 + E1 - D1 + 2);
                                                copy(tour + F1, tour + ncities + 1, tour2 + A1 + E1 - D1 + C1 - B1 +
3);

                                                memcpy(&tour[0], &tour2[0], (ncities + 1) * sizeof(int));
                                                rvereseArray(tour, A1 + E1 - D1 + 2, A1 + E1 - D1 + C1 - B1 + 2);


                                        }
                                        for (j = 0; j < ncities; j++) {
                                                pos[tour[j]] = j;

                                        }

                                        length = 0;

                                        for (j = 0; j < ncities; j++) {
                                                length = length + c[tour[j]][tour[j + 1]];
                                        }

                                }


                        }


                }
                for (j = 0; j < ncities - 1; j++) {
                        Population[r[m]][0] = 0;
                        Population[r[m]][j + 1] = tour[j + 1];
                        Population[r[m]][ncities] = 0;
                }
                Csum_matrix[r[m]] = length;
        }
}
```

```
	double Csum(int arr[])
	{
		int  p1 = 0, p2 = 0;
		double s = 0;
		for (int i = 0; i < ncities - 1; i++)
		{
			p1 = arr[i];
			p2 = arr[i + 1];
			s = s + c[p1][p2];
		}
		s = s + c[p2][0];
		return s;
	}

	void Csum_all()
	{
		int  p1 = 0, p2 = 0;
		double s;

		for (int j = 0; j < N_ind; j++)
		{
			s = 0;
			for (int i = 0; i < ncities - 1; i++)
			{
				p1 = Population[j][i];
				p2 = Population[j][i + 1];
				s = s + c[p1][p2];
			}
			s = s + c[p2][0];
			Csum_matrix[j] = s;
			cout << "Cost of individual_" << j << " = " << s << endl;
		}
	}

};

int main()
{
	cout.precision(9);
	int i, j = 1;
	double k, l;
	vector<double> x, y;

	ifstream file{ "file.txt" };
	file >> i;
	while (i == j)
	{
		file >> k >> l;
		x.push_back(k);
		y.push_back(l);
		file >> i;
		j++;
	}
	file.close();
	cout << "Number of cities = " << x.size() << endl << endl;

	DBMEA ob(x.size());
	ob.Create_cost(x, y);

	double start_s = clock();

	ob.NN();
```

```
        ob.SNN();
        ob.ANN();
        ob.C_group();
        ob.population();
        ob.mutation_coherent();
        ob.mutation_loose();
        ob.local_search();

        double stop_s = clock();
        cout << "Execution time is= " << (stop_s - start_s) / double(CLOCKS_PER_SEC) << " sec" << endl;

        ob.Csum_all();
}
```