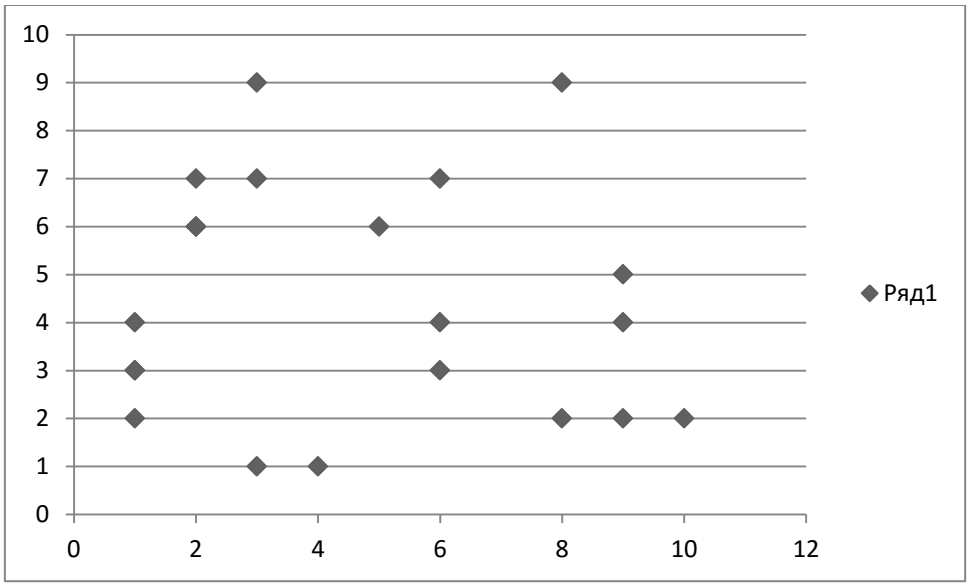
Maxim Tatachka SIIT 05.10.2016

City




```
#include "stdafx.h"
#include <iostream>
#include <windows.h>
#include <vector>
#include <algorithm>
#include <time.h>
#include <math.h>

using namespace std;


// fitness
double fitness(const vector<double>& chromosoma) {
	double summ = 0;
	for (double i = 0; i < chromosoma.size() - 1; i++) {
		summ += chromosoma[i];

		}

	return summ;
}

bool func(const vector<double>& a, const vector<double>& b) {
	return fitness(a) < fitness(b);
}



int main()
{
	srand(time(NULL));
	double sizeOfPopulation, rod2Nomer, min1, min, average, min2, average1,
massXY[40], massMap[20],summ1,x,mel;
	x = 0;
	mel = 9;
	summ1 = 0;
```

```
	min = 0;
	min1 = 0;
	min2 = 0;
	average = 0;
	average1 = 0;
	sizeOfPopulation = 100;

	vector<vector<double>> population;

	// started population
	for (double i = 0; i < sizeOfPopulation; i++) {
		vector<double> chromosoma;

		for (int j = 0; j < 40; j++){
			massXY[j] = rand() % 100 / 10;

		}

		for (int k = 0; k < 20; k++){
			massMap[k] = sqrt((pow(massXY[k]-massXY[k+2], 2) * pow(massXY[k + 1]
- massXY[k + 3],2)));
			cout << massMap[k]<<endl;
			chromosoma.push_back(massMap[k]);
		}



		population.push_back(chromosoma);
	}

	// cout started population
	cout << "Started population:" << endl;
	for (auto e : population) {



		for (auto l : e)

			average += fitness(e);

		if (min1 > fitness(e)){
			min1 = fitness(e);
		}

	}
	cout << "min = " << min1 << endl;
	cout << "AVERAGE = " << average / 200 << endl;
	average = 0;
	for (double steps = 0; steps < 100; steps++) {
		// choose parents and create child
		vector<vector<double>> potomki;
		for (double i = 0; i < sizeOfPopulation; i++) {
			double rod1Nomer = rand() % (population.size()/2);
			vector<double> roditel_1 = population[rod1Nomer];
			double rod2Nomer = rand() % (population.size()/2);
			vector<double> roditel_2 = population[rod2Nomer];
			// create
			vector<double> potomok_1, potomok_2;



			double chislo = rand() % (roditel_1.size() - 2) + 1;

			potomok_1.insert(potomok_1.end(), roditel_1.begin(),
roditel_1.begin() + chislo);
```

```
                potomok_1.insert(potomok_1.end(), roditel_2.begin() + chislo,
roditel_2.end());
                potomok_2.insert(potomok_2.end(), roditel_2.begin(),
roditel_2.begin() + chislo);
                potomok_2.insert(potomok_2.end(), roditel_1.begin() + chislo,
roditel_1.end());
                potomki.push_back(potomok_1);
                potomki.push_back(potomok_2);




        }

        // mutation
        double ver_mutazii = 0.001; // 0.001
        for (double i = 0; i < potomki.size(); i++) {
                double rn = rand() % 100 + 1;
                if (rn < ver_mutazii) {
                        double pos = rand() % (potomki[0].size() - 4) + 2;
                        double buff = potomki[i][pos - 1];
                        potomki[i][pos - 1] = potomki[i][pos + 1];
                        potomki[i][pos + 1] = buff;
                }
        }

        cout << "Iteration: " << steps + 1 <<endl;
        for (auto e : population) {

                cout << fitness(e) << endl;


                for (auto l : e)
                        cout << l << " ";
                        average += fitness(e);

                if (min > fitness(e)){
                        min = fitness(e);
                }
        }

        cout << "min = " << min << endl;
        cout << "AVERAGE = " << average / 400 << endl;


        if ((min == min2) && (average == average1))
        {
                cout << "Finished population:" << endl;
                cout << "min = " << min2 << endl;
                cout << "AVERAGE = " << average / 400 << endl;
                break;
        }

        average1 = average;
        min2 = min;
        average = 0;
        min = 0;


        // select new pop
        vector<vector<double>> vse;
        vse.insert(vse.end(), population.begin(), population.end());
        vse.insert(vse.end(), potomki.begin(), potomki.end());
        sort(vse.begin(), vse.end(), func);
```

```
		population.clear();
		for (double i = 0; i < (sizeOfPopulation); i++) {
			population.push_back(vse.at(i));
		}




		}

		system("Pause");
		return 0;
}
```