Acta et Commentationes, Sciences of Education, nr. 4 (18), 2019 ISSN 1857-0623
p. 96-103 E-ISSN 2587-3636

CZU: 004.02 DOI: 10.36120/2587-3636.v18i4.96-103

A NON-STANDARD METHOD OF SOLVING COMPUTATIONAL GEOMETRY PROBLEMS

Andrei BRAICOV, associate professor, PhD

Tiraspol State University

Summary. Computational geometry problems are often encountered in programming contests. In this article we examine some problems that, traditionally, are solved using vectors. Another approach to finding solutions is proposed, less commonly found in the specialized literature.

Keywords: competitive problems, computational geometry, programming, original solutions.

O METODĂ NON-STANDARD DE SOLUȚIONARE A PROBLEMELOR DE GEOMETRIE COMPUTAȚIONALĂ

Rezumat. Problemele de geometrie computațională se întâlnesc deseori în concursurile de programare. În acest articol sunt examinate câteva probleme care, tradițional, se rezolvă utilizând vectorii. Se propune o altă abordare de căutare a soluțiilor, mai puțin întâlnită în literatura de specialitate.

Cuvinte cheie: probleme competitive, geometrie computațională, programare, soluții originale.

Introduction

Computational geometry problems are often encountered in programming contests at different levels. A series of notions and formulas from analytical geometry are used to solve them.

In the specialty literature there are several standard problems with their solutions: the problem of convex windings; the problem of triangularizations; proximities and belongings etc. [1, 2].

It is obvious that knowledge of analytical geometry (Cartesian coordinates, polar coordinates, the equation of the straight line, coordinate transformations, geometric transformations, vectors, relative positions, intersections, etc.) is required to solve them.

Problems related to the positions of the point relative to another geometric figure are solved starting from the notion of the position of the point relative to a vector, which, for some students, is a difficult subject to understand.

In the following, we will try to describe how to solve some problems of computational geometry related to the positions of the point against another geometric figure without using the notion of vector.

Problem 1. Point and polygon. The natural number n, $n < 100$, the Cartesian coordinates of point M and the Cartesian coordinates of n points A_1, A_2, … A_n, representing the vertices of the convex polygon A_1A_2 … A_n, are given. Determine an algorithm that verifies whether or not the point M belongs to the mentioned polygon.

Solution:

For the point M to belong to the polygon $A_1, A_2, \ldots A_n$ it must:

- coincide with one of the points of the polygon or;
- belong to one of the segments $[A_i, A_{i+1}]$, where $i = 1, 2, \ldots n$ and $A_{n+1} = A_1$.

The point M belongs to the segment $[A_i, A_{i+1}]$ if $d(A_i, M) + d(M, A_{i+1}) = d(A_i, A_{i+1})$. The second case also includes the first.

The complexity of the algorithm is $O(n)$.

Problem 2. Convex polygon. The natural number n, $n < 100$, and the Cartesian coordinates of n points $A_1, A_2, \ldots A_n$, representing the vertices of the polygon $A_1A_2 \ldots A_n$, are given. Write an algorithm that checks whether or not the mentioned polygon is a convex polygon.

Solution:

It is known that the polygon $A_1A_2 \ldots A_n$ is convex only if all its vertices (except for the points A_i and A_{i+1}) belong to the same semiplane determined by any of the line $A_i A_{i+1}$, where $i = 1, 2, \ldots n - 1$.

It can be shown that the requirement in the definition can be "simplified" as follows:

Definition 1. Polygon $A_1A_2 \ldots A_n$ is a convex polygon only if for any two vertices A_j, A_{j+1}, points A_{j-1} and A_{j+2} will belong to the same semiplane determined by the segment $A_j A_{j+1}$, where $j = 1, 2, \ldots, n$, and $A_0 = A_n$, $A_{n+1} = A_1$, $A_{n+2} = A_2$.

In solving the problem we will use the notion of *deviation of a point from a straight line*, a notion less commonly found in the specialized literature.

Definition 2. Let d be a line with the equation $ax + by + c = 0$ and the point $M(m_1, m_2)$. The number $am_1 + bm_2 + c$ is called *the deviation of the point M from the line d*, which we note $Ab(M, d)$.

The following Lemma can be easily demonstrated.

Lemma. Points M and N belong to the same semiplane determined by line d if and only if

$$Ab(M, d) \text{ and } Ab(N, d) \text{ have the same sign. (1)}$$

The relationship (1) can be written as follows:

$$Ab(M, d) \times Ab(N, d) > 0. \text{ (2)}$$

Therefore, taking into account definition 2 and relation (2), we conclude that for the mentioned polygon to be convex the condition must be met:

$$Ab(A_{j-1}, A_j A_{j+1}) \times Ab(A_{j+2}, A_j A_{j+1}) > 0, \text{ (3)}$$

for any $j = 1, 2, \ldots, n$, where $A_0 = A_n$, $A_{n+1} = A_1$ and $A_{n+2} = A_2$.

If we define the type

```
struct t_point {
    double x, y;
} TPoint;
```

then the function for determining the deviation of the point *X* from the line side *YZ* can be defined as follows:

```
double ab(TPoint X, TPoint Y, TPoint Z)
{
 return (Y.y - Z.y) * X.x + (Z.x - Y.x) * X.y + Y.x * Z.y - Y.y * Z.x;
}
```

Thus, the complexity of the algorithm was reduced to O(*n*).

Problem 3. The interior of the polygon [3]. The natural number *n*, $n < 100$, the Cartesian coordinates of point *M* and the Cartesian coordinates of *n* points $A_1, A_2, \ldots An$, representing the vertices of the convex polygon $A_1A_2 \ldots A_n$, are given. Write an algorithm that determines whether or not the point *M* belongs to the interior of mentioned polygon.

Solution:

We will create the solution of the problem using the same notion as in the case of problem 2, i.e. the deviation of a point from a straight line.

In this case must be verified the relationship:

$$Ab(M, A_i A_{i+1}) \times Ab(A_{i+2}, A_i A_{i+1}) > 0,$$

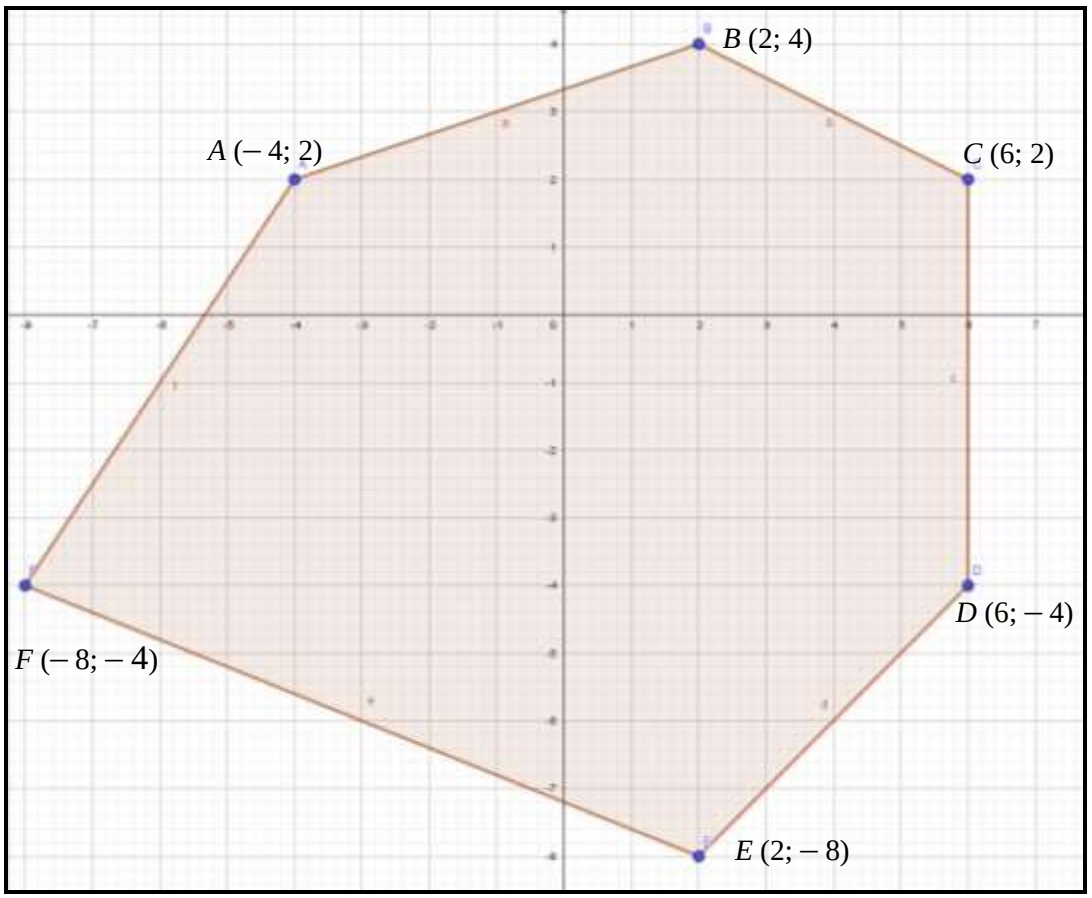
for any $i = 1, 2, \ldots, n$, where $A_{n+1} = A_1$ and $A_{n+2} = A_2$.

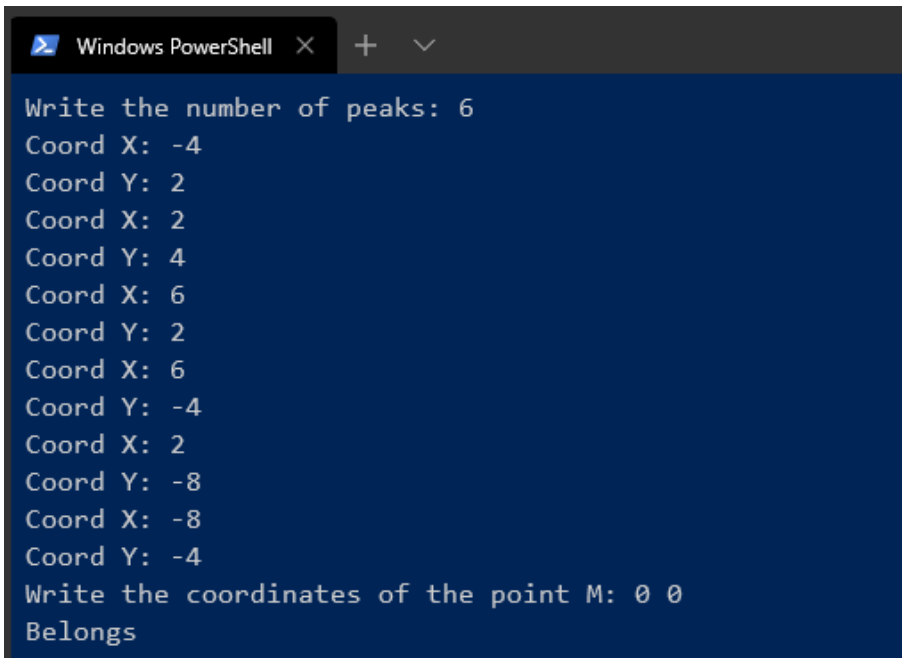
In this case the complexity of the algorithm is also O(*n*).

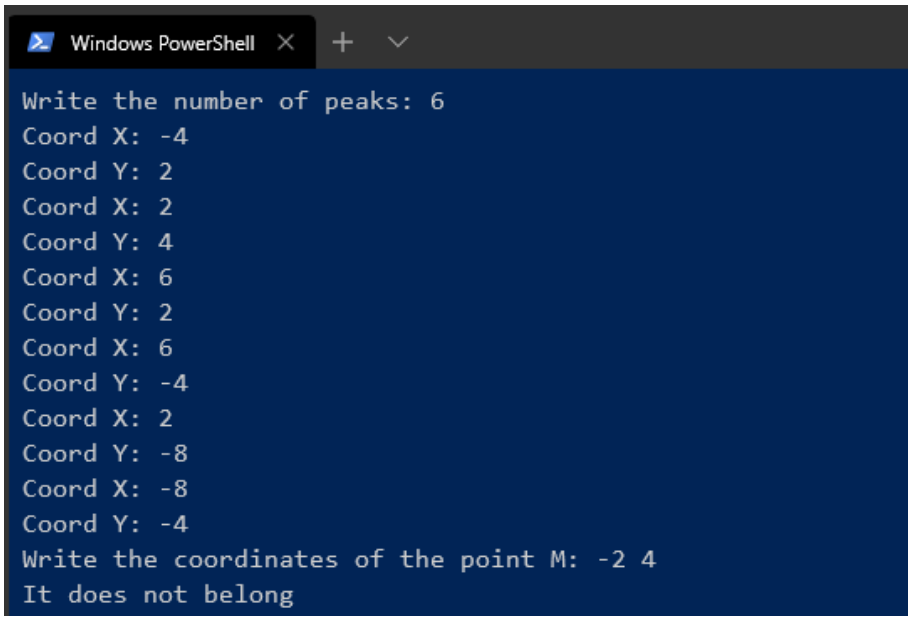
```
#include <iostream>
using namespace std;
typedef struct t_point {
    double x, y;
} TPoint;
//Deviation of a point X from a straight line YZ
double ab(TPoint X, TPoint Y, TPoint Z)
{
 return (Y.y - Z.y) * X.x + (Z.x - Y.x) * X.y + Y.x * Z.y - Y.y * Z.x;
}
int main()
{
    int n;
    TPoint M;
    TPoint *A;
    bool f;
    cout << "Write the number of peaks: ";
    cin >> n;

    A = new TPoint[n + 2];
    for (int i = 0; i < n; ++i) {
        cout << "Coord X: ";
        cin  >> A[i].x;
        cout << "Coord Y: ";
```

```
        cin  >> A[i].y;
    }
    cout << "Write the coordinates of the point M: ";
    cin  >> M.x >> M.y;
    f = true;
    // We add the first 2 vertices to the vector
    A[n].x = A[0].x;
    A[n].y = A[0].y;
    A[n + 1].x = A[1].x;
    A[n + 1].y = A[1].y;
    for (int i = 0; i < n; ++i) {
        if (ab(M, A[i], A[i + 1]) * ab(A[i+2], A[i], A[i+1]) <= 0) {
            f = false;
        }
    }
    if (f) {
        cout << "Belongs";
    } else {
        cout << "It does not belong";
    }
    return 0;
}
```







Problem 4. The outside of the polygon. The natural n, $n < 100$, the Cartesian coordinates of point M and the Cartesian coordinates of n points A_1, A_2, … A_n, representing the vertices of the convex polygon $A_1A_2 \ldots A_n$, are given. Write an algorithm that determines whether or not the point M belongs to the outside of the mentioned polygon.

Solution:

The search for the solution of this problem includes the methods applied to solve problems 1 and 3.

Thus, we will verify that at least one of the two relationships is respected:

1) there is at least one i, where $i = 1, 2, \ldots, n$, and $A_{n+1} = A_1$, $A_{n+2} = A_2$, such that:

$$Ab(M, A_i A_{i+1}) \times Ab(A_{i+2}, A_i A_{i+1}) < 0;$$

2) there is at least one i, where $i = 1, 2, \ldots, n$, $A_{n+1} = A_1$ and $A_{n+2} = A_2$, such that: $Ab(M, A_i A_{i+1}) \times Ab(A_{i+2}, A_i A_{i+1}) = 0$ and $d(A_i, M) + d(M, A_{i+1}) > d(A_i, A_{i+1})$.

In this case the complexity of the algorithm is also $O(n)$.

Problem 5. The area of the polygon. We give the natural number n, $n < 100$, the Cartesian coordinates of n points A_1, A_2, … A_n, representing the vertices of a simple polygon (the edges of the polygon do not intersect). Write an algorithm that calculates the area of the polygon $A_1A_2 \ldots A_n$.

Solution:

a) If $A_1A_2 \ldots A_n$ is a convex polygon (this condition can be verified, see the solution of Problem 2), then the problem will be solved by triangulating the polygon ([1, page 29]), then summing the areas of all the obtained triangles.

b) The problem is more difficult if the polygon $A_1A_2 \ldots A_n$ is concave.

In this case it results that there is at least one i, where $i = 1, 2, \ldots, n$, $A_0 = A_n$, $A_{n+1} = A_1$ and $A_{n+2} = A_2$, that $Ab(A_{i-1}, A_i A_{i+1}) \times Ab(A_{i+2}, A_i A_{i+1}) < 0$. (Figure 1).

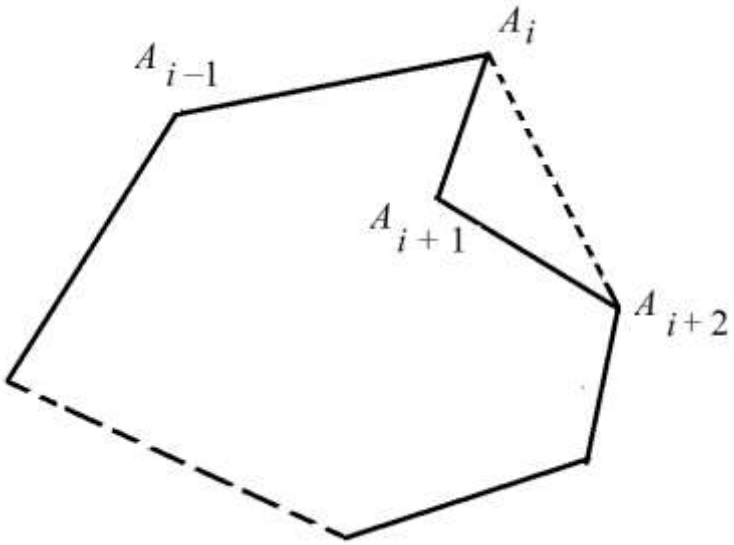


Figure 1. The case of the concave polygon

Figure 1 suggests the following algorithm:

1. "Exclude" the vertex A_{i+1}. The obtained polygon will have the area greater than the polygon $A_1A_2 \ldots A_n$, the difference being equal to the area of the triangle $A_i A_{i+1} A_{i+2}$. We store the area of this triangle in a vector S.

2. We use a "queue" in which we store all the points of the given polygon except the "excluded" vertices. Let $B_1B_2 \ldots B_m$, where $m \leq n$, be the obtained polygon (extracted from "queue").

3. The area of the polygon $A_1A_2 \ldots A_n$ is equal to the area of the polygon $B_1B_2 \ldots B_m$ minus the sum of the elements of the vector S. The polygon $B_1B_2 \ldots B_m$ is convex, so we reduced the problem solving to the case a).

This algorithm is not valid for any polygon!

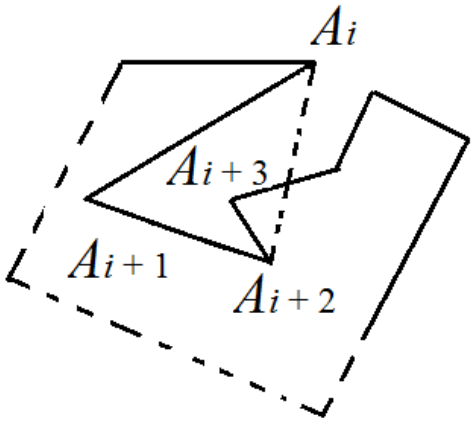


Figure 2. The case 2 of the concave polygon

From figure 2 we notice that if we "exclude" the peak A_{i+1}, then the obtained polygon is not simple, so the described algorithm is not valid.

It can be shown (see, for example, [4, page 258]) that the most "convenient" algorithm for calculating the area of polygon $A_1A_2 \ldots A_n$ is the calculation of the expression value:

$$|0{,}5 \sum_{i=1}^{n}(x_i(y_{i+1} - y_{i-1}))|,$$

where $A_{n+1} = A_1$, $A_0 = A_n$, and x_i and y_i – the coordinates of the point A_i.

Problem 6. The intersection of two segments. The Cartesian coordinates of 4 points P_1, P_2, P_3, P_4 are given. Determine an algorithm that checks if the segments $[P_1P_2]$ and $[P_3P_4]$ intersect.

Solution:

In [4, page 254] the solution of this problem is presented in two stages:

1. *Rapid rejection test*. If the rectangle with the diagonal $[P_1P_2]$ does not intersect with the rectangle with the diagonal $[P_3P_4]$, then neither the segments $[P_1P_2]$ and $[P_3P_4]$ intersect.

2. *Checking of intersection*. If the rejection test is "not passed", then for segments $[P_1P_2]$ and $[P_3P_4]$ to intersect, each of the two segments must intersect the line containing the other segment.

Another way to solve this problem:

1. *Rapid rejection test*. If the lines P_1P_2 and P_3P_4 do not intersect, then neither the segments $[P_1P_2]$ and $[P_3P_4]$ intersect.

2. Let the rejection test be "not passed" and $M = P_1P_2 \cap P_3P_4$. The segments $[P_1P_2]$ and $[P_3P_4]$ intersect if and only if point M belongs to the segment $[P_1P_2]$, that is, only if the relation $(M, P_1) + d(M, P_2) = d(P_1 P_2)$ occurs.

Conclusions

Computational geometry highlights the applicative role of mathematics. In the context of competitive informatics, it strengthens the motivation and the need for a deep study of analytical geometry (in plan and in space).

Obviously, the skills of solving the problems of computational geometry increase the chances of success in the contests and the programming Olympics.

The didactic approaches to study this field must be folded on the degree of understanding of the new concepts by the students. If the use of vectors has difficulties, at the first stage problems should be analyzed for which solutions can be found without vector calculations. Subsequently, one can return to vectors, which represent an efficient tool for solving problems with high degree of difficulty.

Bibliography

1. Corlat S. Metodologia rezolvării problemelor de geometrie computaţională. Chișinău: Tipografia UST, 2013.
2. Ласло М. Вычислительная геометрия и компьютерная графика на C++. Москва: Бином, 1997.
3. Braicov A. Informatică. Turbo Pascal. Culegere de probleme. Editura *Prut Internațional*, 2007. 232 p. ISBN 9975 – 69-788-7.
4. Cerchez E., Șerban M. Programarea în limbajul C/C++ pentru liceu (volumul III). Iași: Editura Polirom, 2006. 296 p.