Computational Geometry - Finding Farthest Pair of Points using Convex Hull and Rotating Calipers Algorithm

Description of Problem

1. Functionality

   Given a set of points in 2D space, find the pair of points with the maximum distance between two.

2. Implementation Requirement

   (a) It is not allowed to use brute force comparison. But the brute force comparison could be used to verify test results.

   (b) It is required to use Convex Hull and Rotating Calipers algorithm to find the farthest pair of points.
   The fundamental insight of this method is that the two farthest points in a set of points must be the vertices on the convex hull, which is the smallest convex polygon enclosing all the points in the set.

   (c) If there exist multiple pairs of points with the exact same maximum distance, find all of them.

3. Input and output

   (a) Input:
   - The first line is an integer n for the number of points.
     The cases for n=1 or 2 are trivial. So the value of n is restricted to be >2.
   - The next n lines give the x,y position of each point

   (b) Output:
   - The first line is the maximum distance square
   - Each of the following lines give the x,y position of each pair of the points with the maximum distance. The format is “(x1,y1) and (x2, y2)”

Solution of Problem

The solution involves 2 steps of processing

1. Find the Convex Hull for a given set of points

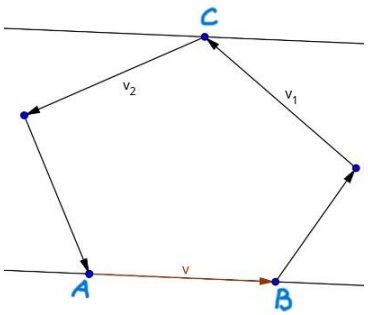
   (a) The Graham Scan algorithm is used to find the convex hull

   (b) The result is a sorted list of vertices (in the counter clockwise order) that form the boundary of the convex hull.

2. Apply rotating calipers algorithm to find the farthest pair of points among the vertices of Convex Hull

   (a) When iterating through the edges of the convex hull in the counter clockwise order, we try to identify the point farthest from each edge.

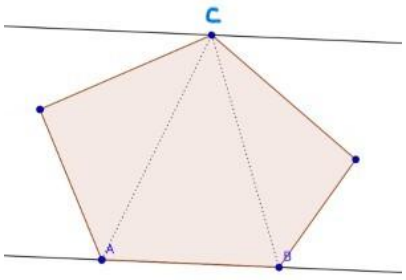
   For the edge $\overline{AB}$ in the following figure, we calculate cross products for $v \times v_1$ and $v \times v_2$. We notice $v \times v_1 > 0$ and $v \times v_2 < 0$. There is a sign change (sign could be +, -, or 0) in the cross product whenever we reach the vertex that allows **a parallel line of support**. Such a vertex is the farthest point from the edge. So $\overline{CA}$ and $\overline{CB}$ are the candidate line segments with the maximum length. The corresponding vertex pairs (C, A) and (C, B) are called **anti-podal pairs**.



   (b) Identify all anti-podal pairs

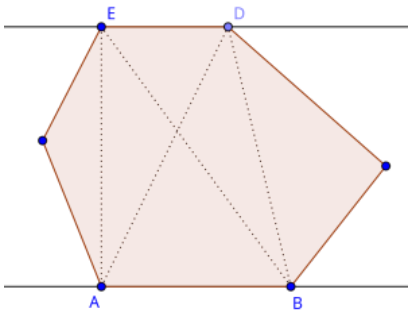
   There are two cases to be considered:

   Case 1: The parallel line of support only intersects a vertex so there are two pairs of anti-podal points. For the following example, C is the farthest point from the edge $\overline{AB}$. The two anti-podal pairs are (C, A) and (C, B)



   Case 2: The parallel line of support intersects an edge of the convex hull so there are four pairs of anti-podal points. For the following example, D is the farthest point from the

edge $\overline{AB}$ and the parallel line of support through vertex D intersect edge $\overline{DE}$. The four anti-podal pairs are (D, A), (D, B), (E,A) and (E,B)



(c) Find the farthest points using anti-podal pairs

The distance square d^2 between two points is calculated When an anti-podal pair is identified.

- $d^2 = (x1 - x2)^2 + (y1 - y2)^2$

The farthest pair is the anti-podal pair with the maximum distance. There may be multiple anti-podal pairs with the exact same maximum distance.

Complexity

The overall time complexity is dominated by the first step of building the convex hull, resulting in O(n log n) time complexity. The space complexity is typically O(n).

Implementation of Solution

The solution is implemented in C++. The file is fp_main.cpp

Test Cases

All the test cases are stored under the directory fp_io.

1. Simple Square test cases (test data file name starting with “simple”)
2. Multiple points test cases (test data file name starting with “small”)
3. Test cases for large set of points (>1000 points) (test data file name starting with “large”)

The brute force comparison is implemented in brute_force_main.cpp. The result (all *.out files) of brute force comparison is used to verify the test result of each test case.