Computational Geometry

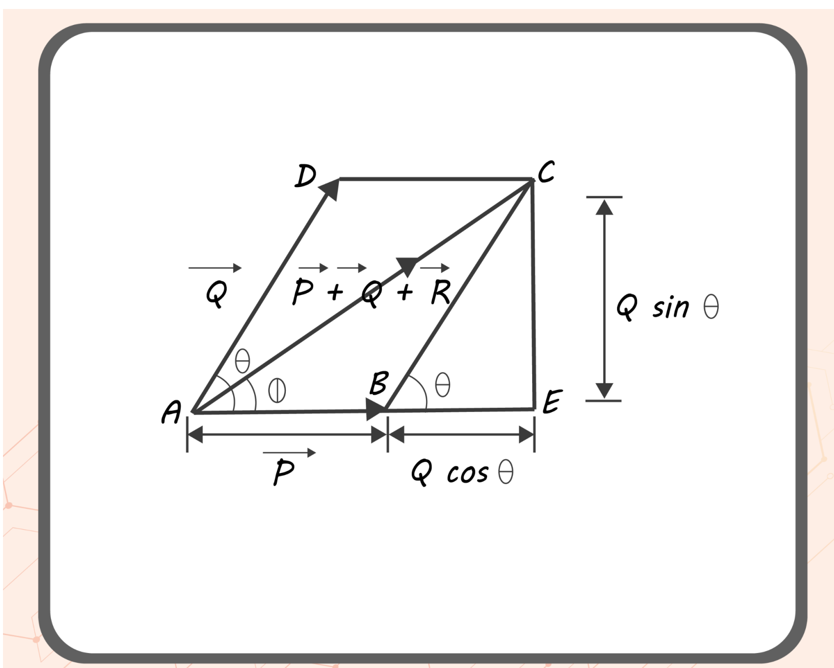
Introduction

To proceed with the applications of geometry in competitive programming, we need to have the basic knowledge of operations like vector addition, dot product, cross product, etc.

Let’s take a look at some of them.

Vector Addition:

According to the parallelogram law of vector addition, if two vectors P and Q are represented by two adjacent sides of a parallelogram both pointing outwards, then the diagonal drawn through the intersection of the two vectors represents the resultant.



the magnitude of the resultant is given by:

R = ($P^2 + Q^2 + 2PQ\cos\Theta$)^{½}

The direction of the resultant vector is determined as follows:

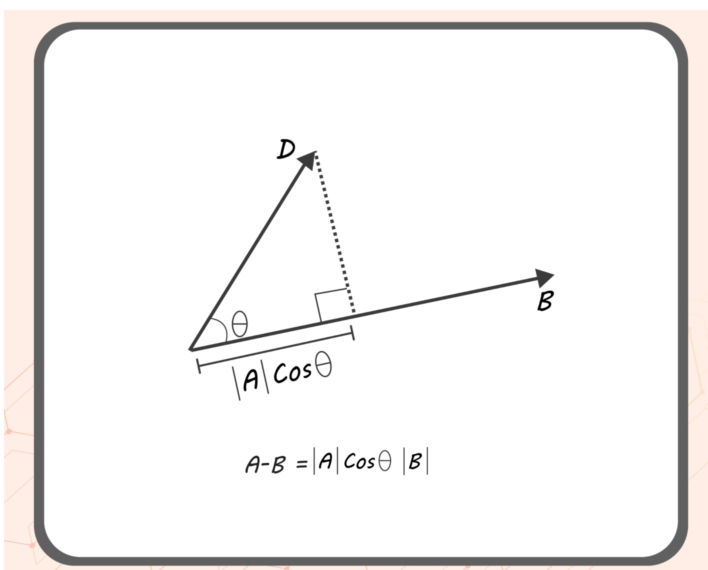
$\tan^{-1}$ ($Q\sin\Theta$ / ($P+Q\cos\Theta$))

Vector Subtraction:

Let us consider that vector a is to be subtracted from vector b. This can also be inferred as the addition of vector a and vector b (in the opposite direction). Thus the formula for addition can be used since (-b) is nothing but b in the reversed direction.

Dot Product:

Geometrically it is a product of the length of the first vector by the length of the projection of the second vector onto the first one.



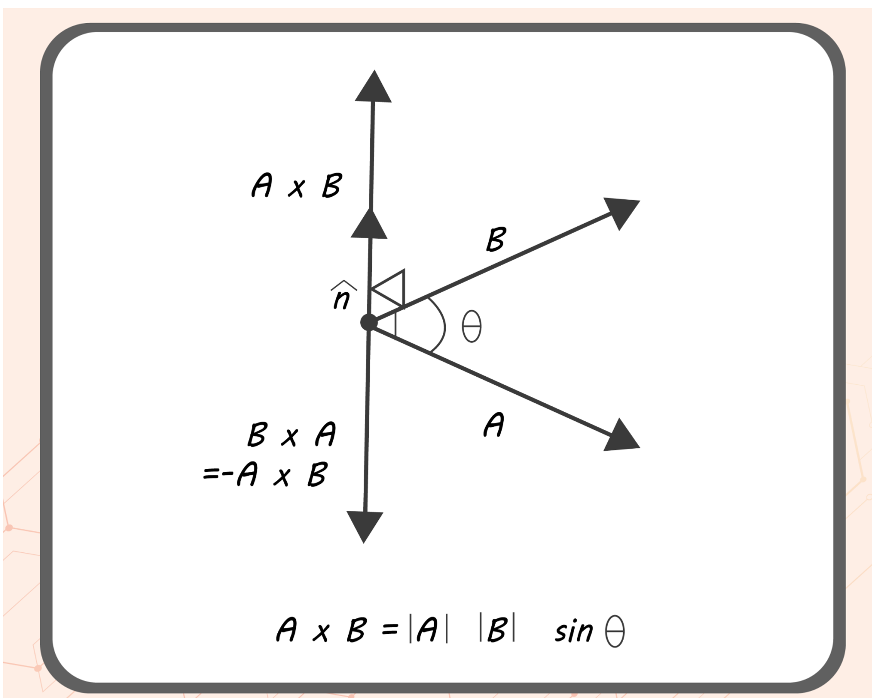
$A \cdot B = |A| \cos\theta \cdot |B|$

The dot product holds some notable properties:

1. A · B = B · A
2. (α · A) · B = α · (A · B)
3. (A+B) · C = A · C+B · C

Cross Product:

The vector product or cross product of two vectors A and B is denoted by A × B, and its resultant vector is perpendicular to the vectors A and B.



A × B = |A| |B| sin θ

Consider two vectors,

A = ai + bj + ck

B = xi + yj + zk

We know that the standard basis vectors i, j, and k satisfy the below-given equalities.

$i \times j = k$ and $j \times i = -k$

$j \times k = i$ and $k \times j = -i$

$k \times i = j$ and $i \times k = -j$

Now, $A \times B = (ai + bj + ck) \times (xi + yj + zk) = ax(i \times i) + ay(i \times j) + az(i \times k) + bx(j \times i) + by(j \times j) + bz(j \times k) + cx(k \times i) + cy(k \times j) + cz(k \times k)$

By applying the above mentioned equalities,

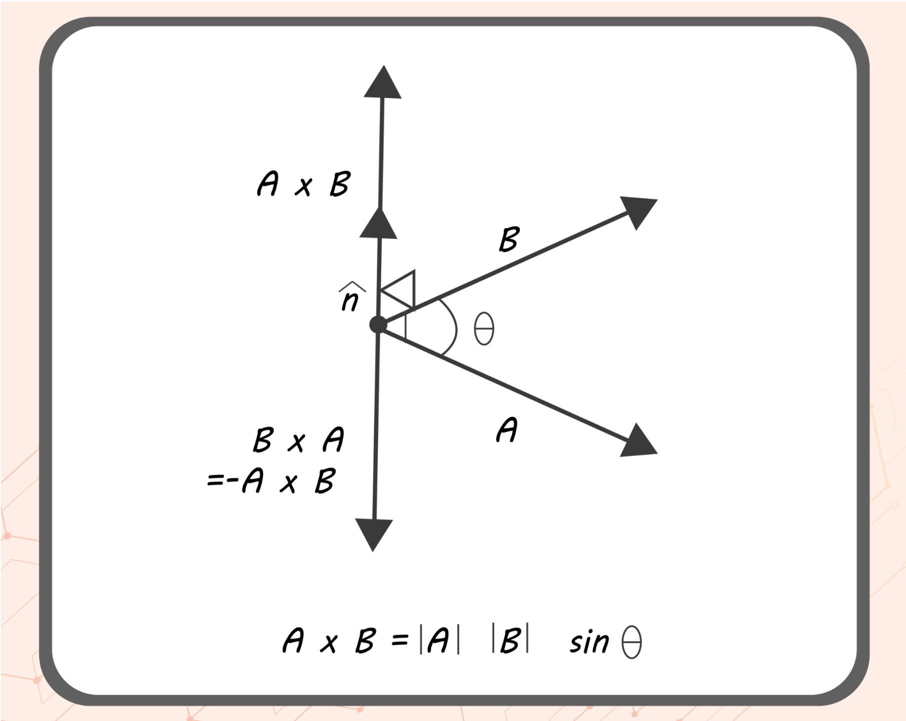
$A \times B = ax(0) + ay(k) + az(-j) + bx(-k) + by(0) + bz(i) + cx(j) + cy(-i) + cz(0)$

$\quad = (bz - cy)i + (cx - az)j + (ay - bx)k$

Distance of a Point

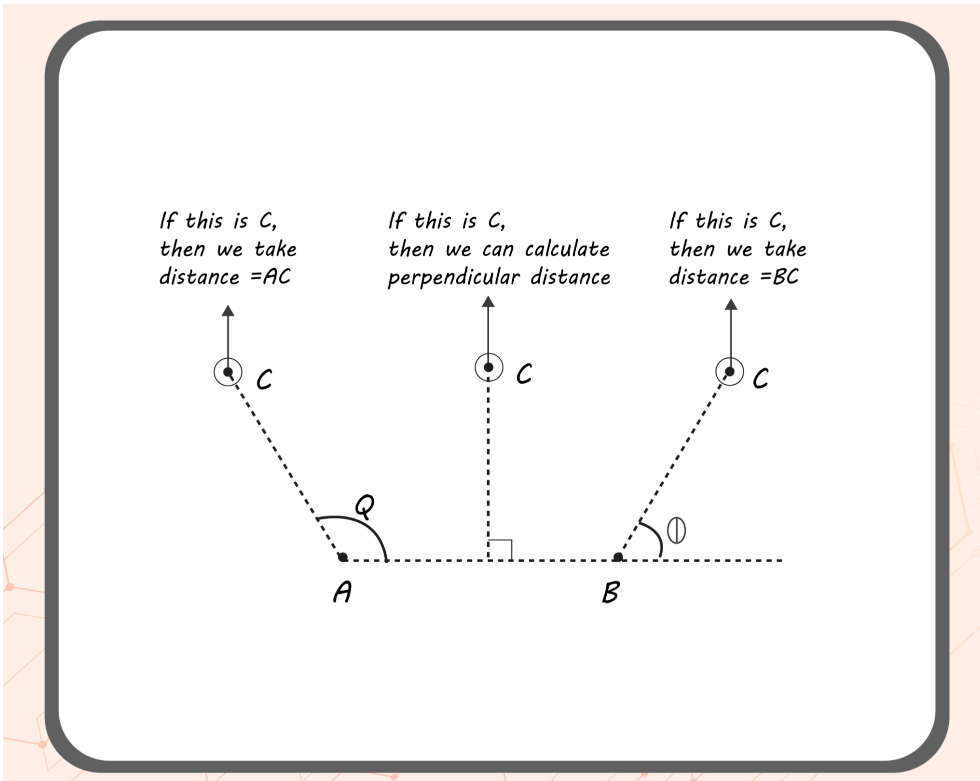
Distance of a point from a line:

Consider a line segment AB and a point C not on the line extending AB. We have to calculate the perpendicular distance of point C from the line.



Distance of a point from a line segment:

If we can’t extend the line segment then there can be three cases in regards to calculating the distance. The position of C can be divided in three categories shown in the figure given below:

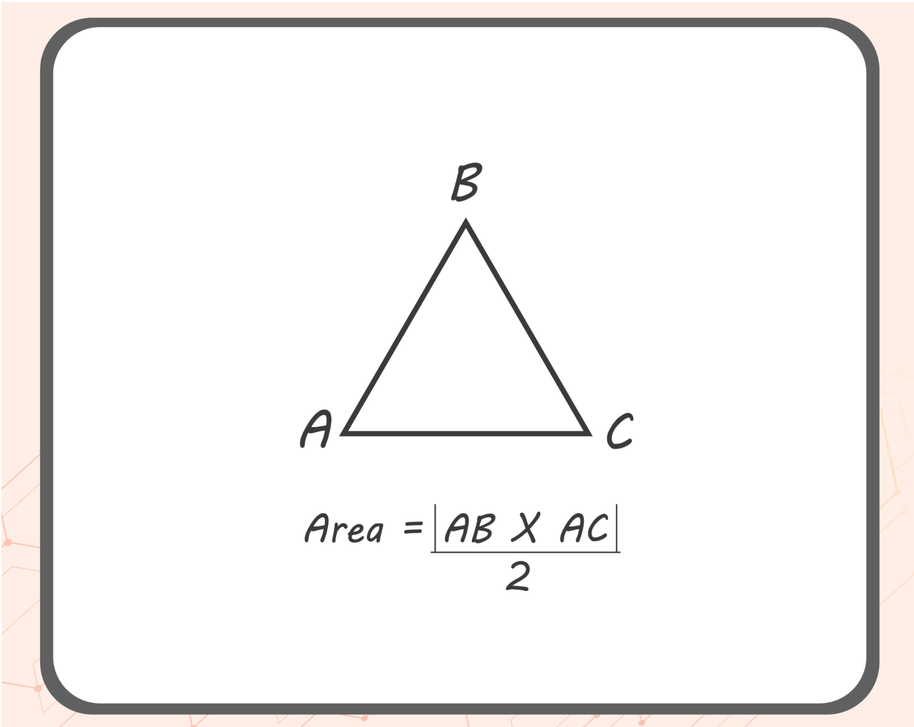


Now, to calculate where C is, we use the concept of the dot product. Let us say the angle between AC and AB is Θ. Then if **AB.AC** is negative, then Θ >= 90°, which means that point C is nearest to pt A on the line segment. If AB.AC is positive, then we check for the angle between AB and BC.

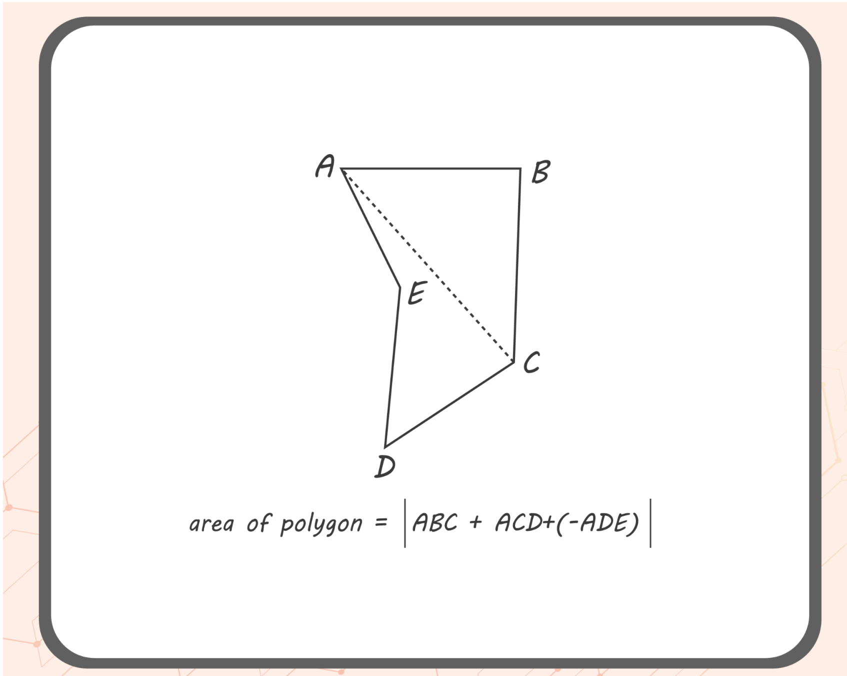
If both the conditions don't hold true then we can calculate the perpendicular distance.

Area of a Polygon

We know that the area of a **triangle ABC** is calculated as **(|ABxAC|)/2**.



This concept/formula can be used to calculate the area of a polygon. We divide the polygon into triangles and the summation of the areas will give us the area of the polygon.



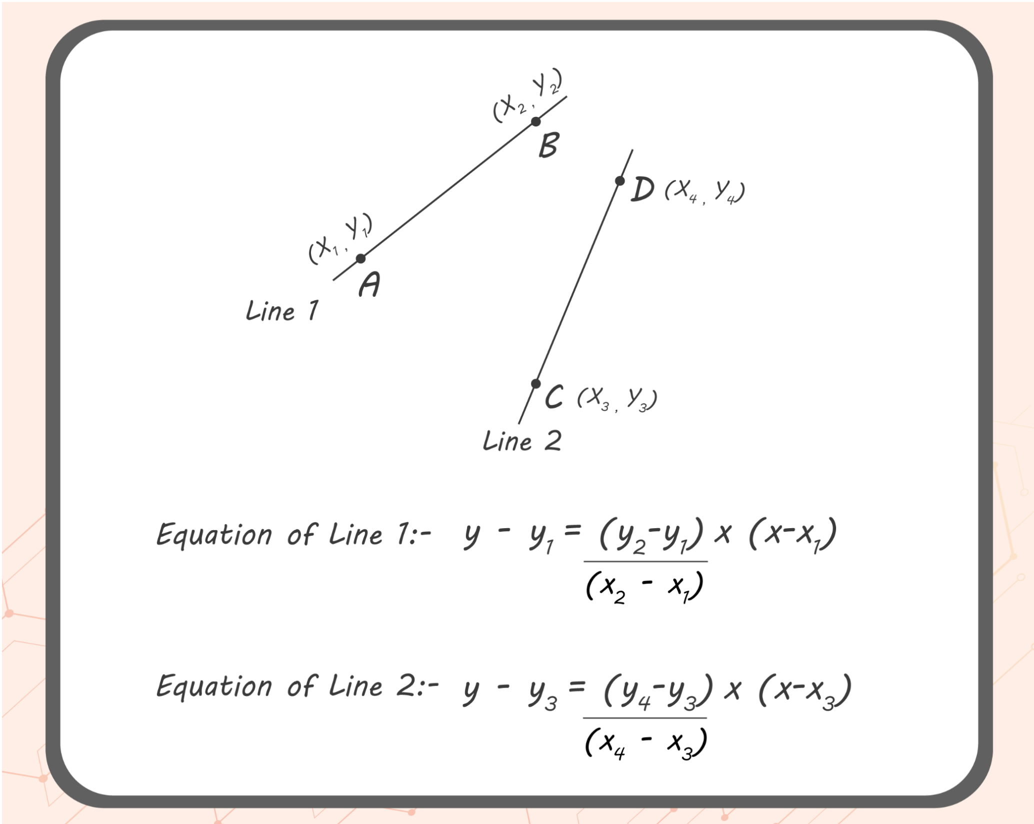
Note: We have to take the cross product because direction of the result also matters since there may be some triangles that form outside the polygon and might need to be subtracted.

Intersection of Two Lines

Problem Statement: We have 4 points A, B, C, D in a 2D plane. Points A and B are on one line and points C and D are on the other. We have to find the point where the two lines intersect each other.

Explanation:

If two points are on a line then the equation of the line can be formed as :



Now, comparing the equation of line 1 with the general equation of line, $\mathbf{a_1x + b_1y = c_1}$ and comparing equation of line 2 with the general equation of line $\mathbf{a_2x + b_2y = c_2}$, we get the values of a_1, a_2, b_1, b_2, c_1, c_2.

$$a_1 = (y_2 - y_1) \qquad a_2 = (y_4 - y_3)$$

$$b_1 = (x_2 - x_1) \qquad b_2 = (x_4 - x_3)$$

$$C_1 = a_1 x_1\ b_1 y_1) \qquad C_2 = a_2 x_3 + \ b_2 y_3$$

Now that we have the variables we need to find the intersection point. Using Cramer's rule:

$$x = -\frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} = -\frac{c_1\ b_2 - C_2 b_1}{a_1\ b_2 - a_2 b_1},$$

$$y = -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} = -\frac{a_1\ c_2 - a_2 c_1}{a_1\ b_2 - a_2 b_1},$$

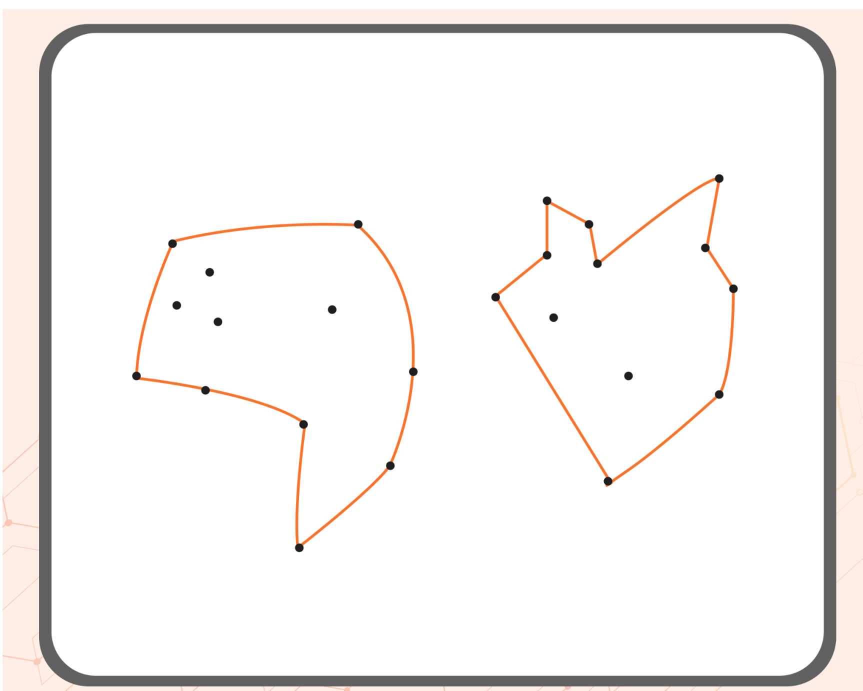
If the denominator equals 0, then either the system has no solutions (the lines are parallel and distinct) or there are infinitely many solutions (the lines overlap). If we need to distinguish these two cases, we have to check if coefficients c are proportional with the same ratio as the coefficients a and b. To do that we only have to calculate the following determinants, and if they both equal 0, the lines overlap.

Convex Hull

Problem Statement: We are given several points on a 2D plane and we have to create a boundary using these points such that no point remains outside and also that the boundary is minimized.

Explanation:

To understand what is meant by the boundary minimization let us take a look at the two possible boundaries that can be formed using the same set of points:



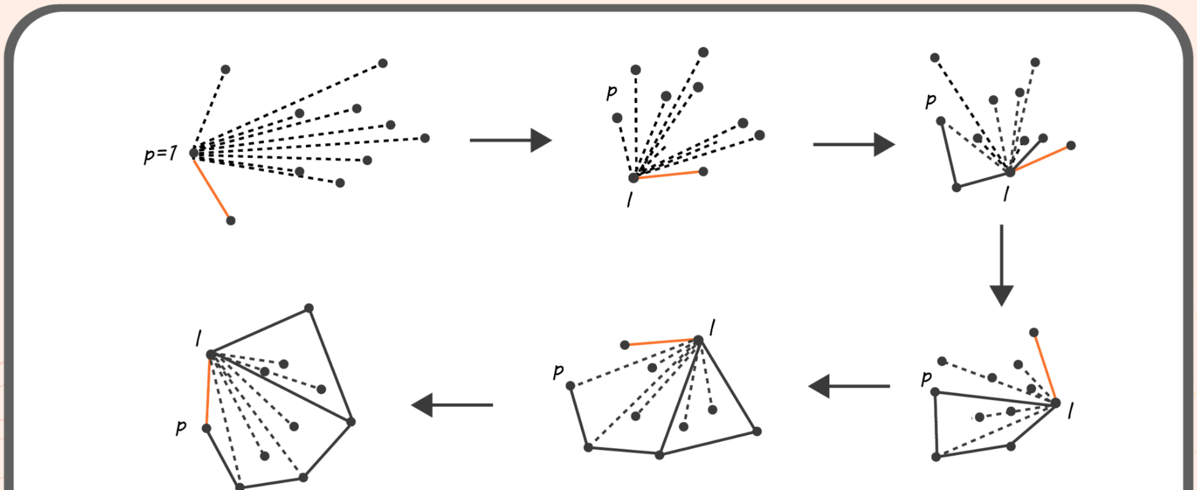
We use the **Jarvis Algorithm** to wrap the given set of points in the convex hull:

1. Initialize p as the leftmost point.
2. Do the following while we don't come back to the first (or leftmost) point.

a. The next point q is the point such that the triplet (p, q, r) is counterclockwise for any other point r. To find this, we simply initialize q as the next point, then we traverse through all points. For any point i, if i is more counterclockwise, i.e., orientation(p, i, q) is counterclockwise, then we update q as i. Our final value of q is going to be the most counterclockwise point.

b. next[p] = q (Store q as next of p in the output convex hull).

c. p = q (Set p as q for next iteration).



The Execution of Jarvis March