Points, Lines, and Vectors

Hello, and welcome to Competitive Programming! Today we are going to start a new unit on computational geometry.

Objectives

After this video, you will have some ideas about where computational geometry problems fall in the context of contests. You'll also review some of the basic formulae for points, lines, and vectors.

Most of the code samples here were cheerfully stolen from the Competitive Programming 3 book.

Contest Strategy

Most contests will have at least one computational geometry problem. Many teams will save that one for last. The reason for this is that such problems often require very tedious coding, and the odds of getting a wrong answer on the first submit are high.

Part of the problem is that there can be tricky edge cases. There can be problems if the test cases gives you things like parallel lines or concave polygons when you weren't expecting them.

Since you are allowed to curate some reference materials at ICPC contests, you should bring your own library of geometry functions instead of trying to memorize things.

Points

Representing Integer Points

Let's get started with points. The easiest way to deal with points is to create a struct with (x) and (y) fields. You'll want two constructors for convenience, and you will also want to overload = and < so that you can do comparisons and sorting. We start with the (x) field and if they are equal we look at the (y) field.

You could also get away with using the STL pair here.

Representing Floating Points

If you want to represent floating points, or maybe I should say (floating point) points, then a pair will not work. The reason is that it's bad to try to compare doubles. Instead, you will need to define a constant EPS. The `fabs` function to take a floating point absolute value can be found in `math.h`.

Formulae

Here are two formulae that you will see frequently. The first is to take the distance between two points. You most likely remember this from high school.

The second formula allows us to rotate a point around the origin. Given an angle (\theta), this matrix will rotate a vector for us. Note that (\theta) is in *radians*, so if you were given degrees you need to multiply by (\pi) and divide by 180.

Lines

Now let's talk about lines.

Representation

There are two ways we can represent a line. The first is to use this equation. The coefficient (b) will be zero if the line is vertical, and one otherwise.

Doing it this way instead of storing the slope directly prevent division by zero errors.

Here is a function that will take two points and return a line for us. We first check that the (x) coordinates are not the same. If they are, we have a vertical line and reply accordingly.

Otherwise, we take the slope by diving delta-y over delta-x for the (a) parameter, and set the b parameter to 1. The (c) parameter gives us the intercept point.

Parallel Lines

Now that we have this, we can easily check if two lines are parallel: if the (a) values are equal and the (b) values are equal. If the (c) values are also equal, then the lines are identical.

Notice again how we never use real equality; we compare with Epsilon instead.

Intersections

Now, if we have two lines, then we can find the intersection point by setting up two linear equations and solving for (x) and (y). It is important to check that the lines are not parallel first, and in the output we need to make sure that one of our lines is not vertical.

Vectors

Representation

A vector is very much like a point, but it represents a direction instead of a location.

Here is a `vec` struct. Be sure when you are communicating with team members that nobody confuses these geometric vectors with standard template library vectors.

In order to convert two points to a vector, you simply have to subtract one point from the other.

We can scale a vector by a factor (s) simply by multiplying it by both fields.

Finally, we can translate a point by a vector by adding the components.

Note that vectors are directions, so we don't talk about translating vectors.

Shortest Distance

There are two operations on lines that yield useful information.

The first is the dot product. The dot product tells you how far one vector goes in the direction of another vector.

If the vector result is zero, then the vectors are at right angles.

Using that, we can write this function dist to line to compute the distance of a point (p) to a line (ab). We first make vectors from (ap) and (ab) and then then take the dot product. We divide this by the square of the distance from a to b to get a parameter u.

This has a fun property: 0 means the intersection was at point a, and 1 means the intersection was at point b. We can then translate (a) along the (ab) line to find the actual intersection point.

Shortest Distance Line Segment

If you have a line segment, the same formula applies, but now you have two special cases: you have to check the end points. If (u) is less than zero, then (a) is the closest point. If (u) is greater than 1, then (b) is the closest point. Otherwise it's the same as the original code.

Angles

You might also need to calculate angles between vectors or lines.

The dot product formula we showed you has another form that involves cosine. So what we can do is take the dot product and then solve for (\theta) using the `arccos` function.

Again, note that the angles are in radians.

Cross Products

The cross product also gives us useful information.

The magnitude of the cross product will tell us the area of the parallelogram induced by the two vectors.

The sign will tell us the direction of the turn. Positive means the second vector leans left, and negative means in leans right.

You may remember seeing this is a physics class.

The functions you see here check if the vectors make a left hand, or counter clockwise, turn; and the collinear function tells us if (r) is on the same line.

Well, that's enough for one video. The next video will go over shapes.

But you should go over all the code that was presented here to make sure you know how it works.

Next
Transcript — Segment Trees →