Finding Polly

We are given lines, defined by point pairs on a small integer lattice, and asked to count how many proper polygons use a segment from each line precisely once.

We can model such a polygon with a permutation of line segments; a pair of adjacent line indices denotes an intersection between two lines. A segment, which is an edge of the polygon, is just a pair of such points.

We can precompute all relevant line intersections.

Enumerate and check, or recurse with backtracking?

Should we just walk through all permutations and check each, or do incremental checking in a recursive solution? We can fix the first line, so we don't need to do 12! permutations but only 11! which is about 40M. But the checking is quadratic, so this probably won't run in time.

Thus, we need to write a recursion that checks each line segment as it is added against other line segments. We know there will be significant early backtracking because most ways to go from line to line quickly lead to line segment intersections.

Numerics

Since each line is defined by a pair of points on the lattice, we have an exact representation of each line. How can we represent a point of intersection? We need to ask if two points are identical (multiple lines can intersect at a given point), and we may need to represent points way outside the given lattice (if two lines are very nearly, but not quite, parallel).

IEEE doubles have 53 bits of mantissa; is this sufficient? Probably? Can we recognize identical points if we use doubles?

Bigints might be too slow . . .

Finding the point of intersection

Given two lines as integer point pairs, what is their intersection? We can represent a line through (x1,y1) and (x2,y2) with the equation:

$$(y2-y1)x + (x1-x2)y = y2\ x1 - x2\ y1$$

Given two lines, we have two equations in two unknowns. This can easily be solved, giving us an equation for each coordinate of the intersection point; this equation is cubic in the numerator and quadratic on the denominator. Thus, the numerator is at most 4000^3 needing 36 bits, and the denominator is at most 4000^2 needing 24 bits. The intended solution was rationals built on 64-bit integers.

Floating Point issues

It is possible to solve this problem using straightforward floating point, and it will probably work if you are extremely careful. But if you decide to use epsilons for comparisons, be very careful; pairs of points created by lines as described in this problem can be exceptionally close (within 5.5e-12 units). For such cases, using an epsilon such as 1e-9 or 1e-10 would probably yield incorrect results.

The judges prepared test cases with such close intersections.