side the polygon (in which case it is a candidate furthest point). Similarly for every edge of the Voronoi diagram we compute all intersections with the polygon edges (and these are also candidate furthest points). This then takes $O(n^2)$ time, because there are $O(n)$ Voronoi vertices and edges, and both point in polygon tests and "compute intersection with every polygon edge" also take $O(n)$ time.

Fun fact: this problem was originally proposed with much larger polygons, requiring $O(n \log n)$ time, but we felt that this was just too hard, even for the World Finals. In this version, computing the Voronoi diagram in $O(n \log n)$ time is actually *not* the hardest part – the second part of finding the furthest point in the Voronoi diagram is even harder (it can be done using a complicated sweepline algorithm, but one has to be very careful, because the number of intersections between polygon edges and Voronoi edges can be quadratically large, so even enumerating all of them would be too costly).

Another approach than the one described above is to do a binary search on the answer and then try to determine efficiently whether a given circle radius covers the polygon or not. That task can be done in $O(n^2 \log n)$ with a sweepline approach. We also saw at least one team solving the problem with a direct sweepline approach without binary search or explicitly computing the Voronoi diagram, but our impression was that this solution was in some sense implicitly computing the Voronoi diagram and incorporated the second part into it.

## Problem H: Single Cut of Failure

*Shortest judge solution: 1484 bytes. Shortest team solution (during contest): 1519 bytes.*

At first glance this looks like a very hard geometry problem, but a moment's thought reveals that the problem is more combinatorial than geometric in nature. Linearizing the coordinates, we see that what we get are a bunch of intervals (on a circle) and we need to pick the smallest number of new intervals such that each input interval crosses at least one of the constructed intervals. This still looks pretty hard though, and the key observation is that the geometry of the problem actually should not be ignored completely: the fact that all wires connect two different sides of the door means that it is always possible to cut all wires using two diagonal cuts.

This means that the problem now reduces to figuring out if we can cut all wires using a single cut or not (hence the problem name). If not, using the diagonals is an optimal cut. Checking if a single cut suffices can be done in linear time after sorting all the wire end points. We keep two pointers $s$ and $t$ (indicating that the cut we make is from $s$ to $t$), initially pointing to the same position. Then, we repeat the following, until the $s$ pointer has gone a whole lap around the date:

1. while we can advance $t$ without making any wire contained within the interval $(s, t)$, do so.
2. advance the $s$ pointer.

(Here, "advance" means moving the pointer past the next wire end point.) During this process we keep track for each wire whether it is completely outside, partially inside, or completely inside the interval $(s, t)$ and keep a count of how many intervals are partially inside. If at any point during the process all $n$ wires are partially inside, we have found our single cut.

One small caveat is to make sure that the cut we make goes between two different sides of the door.