Code Jam 2018 - World Finals

Analysis: The Cartesian Job

The first thing to notice about the problem is that all lasers rotate at the same speed. That means that after each second, regardless of direction, the lasers will be back at the configuration given in the input, and each subsequent second will just be a copy of the previous one. Then, we only need to check what happens within a second. In what follows, we assume the configuration given in the input happens at time = 0 seconds and we only care about what happens for times in the range 0 to 1 seconds.

Test set 1

In Test set 1 there are such a low number of lasers that we can try every possibly combination of directions and add $2^{-\mathbf{N}}$ to the result for each one that leaves the segment uncovered some of the time.

For fixed directions of rotation, we can check whether the segment is covered by first mapping each laser to the interval of time during which it will cover the segment and then seeing if the union of all those intervals covers the full second. We refer to intervals in the modulo 1-second ring; that is, an interval can start at t_0 and end at $t_1 < t_0$, covering the time from t_0 to 1 second and from 0 seconds to t_1. There are several ways to check if the union of intervals, even intervals in a ring, cover the full second, but a simple one is to split the intervals that wrap around into two non-wrapping-around ones, sort those intervals by start time, let t = 0, and do for each interval (t_0, t_1), if $t_0 > t$, then there's a hole; else, $t = \max(t, t_1)$. If we reach the end, there's a hole if and only if $t < 1$.

To obtain the covering interval for a given laser with endpoint p and second point q we check the angle qp(0,0) and qp(0,1000), and then divide by 2π. Notice that if we do this with floating point arithmetic, it is highly likely that we will have precision problems. We can do it with integer only arithmetic by representing the times symbolically as the angles represented by the original vectors. To simplify the presentation, we assume we computed actual times, but all the operations needed for the rest of the solution can be implemented for an indirect representation of those times intervals.

Since we need to check every combination of clockwise and counterclockwise rotations, and evaluating a particular combination requires looping over each laser, this solution takes $O(\mathbf{N} \times 2^{\mathbf{N}})$ time, which is sufficient for Test set 1.

Test set 2

We start by observing that the two time intervals corresponding to each direction of rotation of a given laser are symmetrical; that is, they are of the form (t_0, t_1) and $(1\text{-}t_1, 1\text{-}t_0)$. Notice that the symmetry is both to the 1/2 second point and to the extreme 0 seconds / 1 second, because we are in a ring. Adding the fact that intervals are less than 1/2 long, we can notice that the pair of intervals coming from a given laser can be categorized into one of three types:

- Two non-overlapping intervals, one within (0, 1/2) and the other within (1/2, 1). Example: [0.2, 0.3] and [0.7, 0.8].
- Two intervals that overlap around 1/2. Example: [0.3, 0.6] and [0.4, 0.7].
- Two intervals that overlap around 0 a.k.a. 1. Example: [0.8, 0.1] and [0.9, 0.2].

For the last two types, the overlapped area is an interval of time during which the segment is guaranteed to be guarded. Therefore, we can remove the overlapped part from further consideration and assume that the two intervals are just their symmetric difference. In the first example above, we'd remove [0.4, 0.6] from consideration and keep the pair of intervals [0.3, 0.4] and [0.6, 0.7]. After we do this with all intervals, the part that remains to be considered is some subinterval of (0, 1/2) and some subinterval of (1/2, 1), and each pair of intervals is exactly two symmetrical intervals, one on each side.

At this point the problem we need to solve is, given two symmetrical intervals u_1 within (0, 1/2) and u_2 within (1/2, 1), and a set of pairs of intervals (a, b), (1 - b, 1 - a), what is the probability that the union of picking one from every pair uniformly at random does not cover both u_1 and u_2? Notice that because of symmetry, one particular split (a collection of one interval from each pair) covers u_1 if and only if the opposite split covers u_2. So, an alternate way to frame the problem is: given the list of intervals from each pair that are inside u_1, what is the probability at least one part of a random split of them doesn't cover u_1? This problem we can solve with a dynamic programming approach over the list of intervals.

The dynamic programming algorithm is, in a way, a simulation of the algorithm we presented above to check if the union of the intervals covers the universe, over all combinations at once. We iterate the intervals in non-decreasing order of left end. We also keep a state of which prefix of u_1 each side of the split has already covered. Formally, if $u_1 = (v, w)$ we calculate a function f(i, x, y) = probability of a split of intervals i, i+1, ..., **N** not covering *both* the remainder of the first side (x, w) *and* the remainder of the second side (y, w). Equivalently, this is the probability of the cover not being full given that intervals 1, 2,..., i are split in a way that the union of the intervals from one side of the split is (v, x) and the union of the intervals from the other side is (v, y). The base case is if min(x, y) ≥ w, then f(i, x, y) = 0, or if i > **N** or min(x, y) > the left end of the i-th interval, then f(i, x, y) = 1. For all other cases, we can calculate f(i, x, y) = (f(i+1, max(x, b), y) + f(i+1, x, max(y, b))) / 2, where (a, b) is the i-th interval in the order. Finally, the answer to the problem is f(0, v, v).

Noticing the values x and y for which we need to calculate f are always either s or the right end of an interval bounds the size of the part of f's domain that we need, and thus the running time of the algorithm, by $O(\mathbf{N}^3)$. We can further notice that max(x, y) is always equal to the maximum between s and the maximum right end of intervals 0, 1, ..., i-1. This reduces the domain size and running time to $O(\mathbf{N}^2)$. One further optimization needed is to notice that if min(x, y) is not one of the largest K right ends of intervals 0, 1, ..., i-1, then the result of f(i, x, y) is multiplied by 2^{-K} or less to calculate the final answer. For values as small as 50 for K, that means the impact in the answer of the value of f is negligible in those cases and we can just approximate it as 0, making the size of the domain of f we have to recursively calculate only O(K × **N**).

Implementing the dynamic programming algorithm as explained above can be tricky, especially if you want to memoize using integer indices over the intervals and largest K last values, as opposed to floating point numbers. However, doing a forward-looking iterative algorithm can be a lot easier. We maintain a dictionary of states to probability, where a state is just the two values x and y, always sorted so that x ≤ y. We start with just {(s, s): 1}. The, we consider each interval (a, b) iterative and for each state (s_1, s_2) with probability p in the last step's result, we add probability p / 2 to new states sorted(max(s_1, b), s_2) and sorted(s_1, max(s_2, b)) if a ≤ s_1. If a > s_1, we add p to our accumulated result and don't generate any new state, since state (s_1, s_2) is guaranteed to leave some unguarded time. This is a simple implementation of the quadratic version, but the real trick is when making the optimization to bring the time down to linear, which in this case is simply ignoring states with too low probability (i.e., if p < ε, do nothing).