Code Jam 2015 - World Finals

Analysis: Crane Truck

In this problem, we are given a version of a small programming language, and we are required to track the number of operations executed as a program runs. The memory is a circular array of 2^{40} positions, and all values are considered modulo 256 (for convenience, we shift values one position to make them go [0,255]). The program can run for a really long time, so our approach is to simulate it efficiently by bundling together many of those operations. The actual code is long and complicated — this is the last problem in a finals round, after all — so we just present an overview of the main idea.

The program can be factorized into 5 parts: A(B)C(D)E. Some of these may be empty; in particular, in the Small dataset, D and E are always empty. Each sequence of simple instructions can be translated, via simulation, into a complex instruction of the form:

1. Let i be the current position of the truck.
2. Add some x_j to position i+j for j in [-2000, 2000]. (2000 is the maximum length of each part.)
3. Move the current position to i+k, where k is also in [-2000, 2000].

Running A
After executing A, only values within 2000 of the starting position can be modified. Let us call the area within 2000 of the starting position the "red" area.

Running (B)
After running B several times, we can either finish or move out of the red area. After several more runs outside, the memory outside the red area will start to look periodic with period |k| and we are going to keep moving the current position according to B's k in the 3rd step until we come back to the red area on the other side, completing a full circle. So, we bundle together all the instructions required to complete the circle and we represent the non-red area as a repetition of a period with length at most 4001 (the range of j's in step 2). We can keep doing this in this way until the loop finishes (which is guaranteed). Notice that after at most 4001 complete circles around the memory, we are going to repeat the initial positions inside the red area. And once we've done that, we are going to start repeating the values there, so we can't do more than 4001 × 256 loops without it being an infinite loop (which is forbidden by input constraints).

Running C
After that, we can simulate C, which can modify the range [-4000, 4000] because the previous loop finished within 2000 of the starting point. Let us call that the "green" area.

Running (D)
When running D, something similar to running B happens, but the memory outside the green area is now the sum of two periodic things with period of size at most 4001, so it can have a period of up to 4001^2. Luckily, this is still small enough to simulate, and other than that, we can proceed with the same idea as we did for B.

Running E
For E, we proceed again in the same fashion. In this case, the non-periodic area's size can increase to a couple of millions due to the quadratic size of the last period, but that is still well under the memory limits.