MIKRO

One way to determine if and when two bacteria will meet is to geometrically represent their paths as lines and intersect those lines. Luckily, the discrete nature of the bacteria's movement allows for a simpler approach. For a pair of bacteria, we check if they will be closer after one second. If so, it is possible that they will meet and it is possible to calculate the time from their initial distance and the speed at which they are getting closer (which is constant). An additional check is needed to see if the bacteria really meet at the calculated time, because it is possible that they are initially moving closer, but miss each other and never meet.

We can find the meeting of the highest rank by examining pairwise meetings of all bacteria and find the one that occurs most frequently. Each meeting is uniquely described by the triplet (x, y, t). If R bacteria participate in a meeting, then it will appear $R \cdot (R-1)/2$ times in the list of pairwise meetings (each pair of bacteria participating will contribute one occurrence).

The complexity of this approach is $O(N^2 \log N)$ if we find all meetings and sort them. This can be reduced to $O(N^2)$ with a hash table.

An even simpler and more efficient approach is this: for each bacterium A we find all of its meetings with other bacteria. When we determine that bacterium A meets another bacterium at time T, we check if bacterium A has already met a third bacterium at that time, and increase the rank of that meeting if so.

A problem arises – how do we maintain the meetings and their ranks? Using a binary search tree would reintroduce a logarithmic factor. An array of length depending on the time of meeting (can be as much as 2000001 in this task) would need reinitializing for every bacterium A, which would be too slow. This last deficiency can be fixed using so-called cookies, where every new bacterium A treats an element of the array as uninitialised until it has set its cookie in it.

YOUTUBE

Let next(F, n) be the n-th movie after some film F. We are given next(F, 1) for each F. Is is possible to easily calculate next(F, 2^p) for each F and p as next(F, 2^p) = next(next(F, 2^{p-1}), 2^{p-1}).

From the binary representation of M we can easily calculate next(F, M) using pre-calculated shifts for powers of 2. The overall time complexity is O(N log M + K log M).

By carefully ordering the operations, the model solution does not need to keep the entire precomputed table in memory, but only for adjacent powers of 2. The memory complexity is O(N + M).

STAKOR

The basic idea behind the solution is dynamic programming. Let f(n, pos) be the smallest number of times the rat needs to move, if after the box moves n times, it is at position pos. The rat's possible next moves are limited by the position of the box (which is known). The next position of the box is also know so the rat needs to move to the appropriate edge of the box (for example, if we know the box must move up, the rat must move to one of the cells in the top row of the box and then up again to push the box). The dynamic programming relation is:

f(n, pos) = min{ distance(pos, pos') + f(n+1, pos') } for each possible new position pos'

There are several issues with a direct implementation of this idea.

The first is the memory usage because of the large number of states. The parameter n can be up to a million and the position can be anywhere in the cage. The memory usage can be reduced to just a couple hundred bytes if we calculate the values of f in decreasing order of n, keep only two rows of the DP table, and recognize that the possible positions of the rat are limited to K cells (one edge of the box, depending on where it last moved).

Another problem is the time complexity of calculating distances; doing so as we calculate the values of function f in $O(K^2)$ will be too slow. However, preprocessing the distances between all K^2 cells of a box and all four edges of the box, for each of the $(N-K)^2$ possible positions of the box, will be efficient enough. This last part must also be carefully implemented so that it doesn't use too much memory.