Code Jam 2018 - World Finals

Analysis: Go, Gophers!

Let M = 25 be the maximum possible number of gophers.

Test set 1

Our approach to test set 1 will be as follows: first, we will find the minimum taste level among all gophers, and then we will use it to determine the total number of gophers N.

To find the minimum taste level, we can simply binary search using the question "is the minimum taste level strictly greater than X?" This is equivalent to asking: "Would a snack of quality X go uneaten by all gophers?" To answer such a question, we can offer 2M-1 snacks of quality X consecutively, which guarantees that each gopher is exposed to at least one of them. If none of those snacks are eaten, every gopher's taste level must be greater than X, but if at least one snack is eaten, then there is at least one gopher with taste level at or below X. We can even stop offering snacks as soon as one gets eaten, if we want to save some snacks. This requires ceil(log 10^6) × 49 = 980 snacks at the most.

Once we know that there is a gopher g with level exactly L, and that L is the minimum taste level, we can use our snacks to answer a query "is it g's turn?" by offering a snack of quality L, because g is the only gopher who would eat it. If we offer that many times in a row and calculate the fraction of eaten snacks, that should approximate 1/number of gophers fairly well. At this point, we might as well use our enormous number of leftover snacks to estimate, and then just answer with the N such that the result is closest to 1/N. It turns out that M^3 tries of this experiment guarantee that the answer is correct, even in the worst case. The proof is included below as part of the explanation of the solution for test set 2.

Test set 2

The solution from test set 1 does not extend naturally to test set 2. In particular, it no longer suffices to find out what fraction of gophers have the minimum (or maximum) taste level, because there could be multiple gophers with that taste level; if we find a fraction of 1/K, the number of gophers could be any multiple of K. So, we need to investigate other levels. Investigating taste levels other than an extreme one brings about the problem of the result being impacted by gophers with taste levels other than the one we are interested in. We can still use the idea of answering general queries by repeating snacks of the same quality, but they are significantly more complicated than a simple disjunction of the snack outcomes.

The first query Q we describe is somewhat natural: What is the exact fraction of gophers with taste level ≥ X?

Notice that this query alone is enough to solve the problem: we can do a binary search of sorts: given a range [A, B] of at least two levels, and known fractions of gophers of those taste levels X and Y, respectively, we can calculate the fraction of gophers Z for taste level C = (A + B) / 2. We recurse over the range [A,C] if X ≠ Z and over the range [C,B] if Y ≠ Z, because the fractions are different if and only if there is a gopher with a taste level in the range producing the change. This algorithm allows to identify all levels at which at least one gopher exists. For each of them we calculate the fraction of gophers at level L using our query and then subtracting the fraction of gophers at each other level < L. Finally, we can take the least common multiple (LCM) of the denominators of all those fractions to find the answer.

This algorithm requires about N × ceil(log 10^6 - N) queries like Q to be made. Unfortunately, we see below that this would be too many.

One way W1 of solving Q is to try enough snacks and then round to the nearest feasible fraction. If we use enough snacks, the rounding will give the exact answer. Note that if we give X consecutive snacks, we are guaranteed to have the right answer in all but a prefix and a suffix of the tries, both of which have length up to M-1. This bounds the error by 2M-2. Moreover, since our experiment has a binary answer, the farthest we can be from the true number of positive (or negative) answers is M-1, in the case when there are exactly ceil(M/2) answers of one type and floor(M/2) of the other, and we happen to hit both a prefix and suffix of size floor(M/2) giving the same answer (this is for odd M, the worst case would be M for an even M).

Additionally, since we only consider fractions with denominators up to M, the distance between two different results is bounded by 1 / (M × (M-1)). This means that if our experiment has an error of less than half of that, rounding is guaranteed to produce the right answer. Putting the total error of M together, this implies that we need M^3 snacks to get a perfect answer for Q. A total number of snacks of M × ceil(log 10^6 - M) × M^3 exceeds the allotted total by a large margin.

A different way W2 to answer Q is to always use R consecutive snacks, where R is the LCM of all the possible results. That means the error is always zero, since we give each gopher exactly the same number of snacks. Unfortunately, LCM(2,3,...,25) is also much too large, so this doesn't work either.

Another strategy to reduce the number of queries is to notice that we only need an exact number for the final levels, right before doing the LCM to get the result. For all intermediate values of our "multi-leaf binary search", we only need to decide whether there is some gopher in a range or not. One way W3 to do it would be to, instead of using exact fractions X, Y and Z for A, B and C above, have approximations X', Y' and Z' that are good enough that we can decide whether the real numbers are equal. Using $2M^2$ snacks guarantees that we will encounter each gopher at least 2M times, which means that if X and Z are different, their approximations of the number of total positives will differ by at least 2M. Since we showed that the total error of both approximations is at most M-1, the error of the difference is at most 2M-1, which means comparing that difference with 2M is enough to determine whether the real fractions are equal or not. This significantly reduces the total required number of snacks, since we only need to use M × ceil(log 10^6 - M) × M^2 for the multi-leaf binary search, and then M × M^3 or M × R to get the final precise answers. However, this is still too many.

The final algorithm requires us to use all 3 of the variants above: we start the multi-leaf binary search using W3. Each time we find a level, we use either W1 or W2, whichever is cheapest, to get the real fraction of the found level. If we recurse on the larger intervals first, we'll find the levels from highest to lowest, so we can do the subtraction. Once we have the real fraction, we potentially restrict the number of possible results N to the multiples of the denominator. This reduces R. Eventually, R might be small enough that we can use W2 for the binary search instead of W3 as well. Notice that each time we use W2, since we started with other methods, we need an additional R to "align" ourselves, depending on how many snacks we have used so far. However, since we eventually use R for everything, the additional cost of these alignments is extremely small.

We leave the precise analysis of total number of snacks needed to the reader, but it's possible, with some careful bounding, to prove that it never exceeds **S**, and we couldn't find a case that gets above ~85% of **S**. The reason is that if the denominator of a fraction is larger than M/2, then there's only one possible result left and we are done. If it's less than M/2 but somewhat large, the number of possibilities for the result is reduced significantly, and its LCM is reduced even more because the GCD of those possibilities is more than 1. If the denominator is small, it means that the found level has multiple gophers, which reduces the total cost of the multi-leaf binary search.