Code Jam 2021 - Round 1A

Analysis: Hacked Exam

In this problem we want to maximize our expected score. Let p_q be the probability of question q's answer being `T`, and let $Q_{\texttt{T}}$ and $Q_{\texttt{F}}$ be the sets of questions we answer with `T` and `F`, respectively. By linearity of expectation, the expected score to maximize can be written as

$$\left(\sum_{q \in Q_{\texttt{T}}} p_q\right) + \sum_{q \in Q_{\texttt{F}}} (1 - p_q) \, .$$

Notice that linearity of expectation can be used regardless of any conditional probability between different questions. In this case, those conditional probabilities can be quite complicated, so being able to ignore them and treat each question independently despite them not being independent in the probability theory sense simplifies the solution process significantly. This is a trick that is important in many problems about expected values.

We operate with fractions throughout most of the proposed solutions. As noted in the sample, the needed numbers may exceed 64 bits for Test Set 3, and potentially for Test Set 2 as well, depending on the exact implementation of the algorithm and the fraction operations. It is absolutely possible to solve Test Set 2 with simply 64-bit integers and Test Set 3 with just 128-bit integers. Our most popular languages all support this in some way: C and C++ have `__int128` support, Java and C# have `BigInteger`, JavaScript has `BigInt`, Bash has `bc`, and Python, Ruby and Haskell have native long integer support. We usually strive to make most of our problems solvable with just 64-bit integer arithmetic, but in this case, limiting the number of questions so much would allow suboptimal solutions in fast languages to pass Test Set 3.

Notice that we can have information of 1 or 2 other students in the first two test sets, and also 3 in the last test set. We could solve each number of students independently, but there is no need for that. Adding a student with the same answers and score as a student in the input results in a completely equivalent case, so we can always assume that there are a maximum number of students by copying any student the needed number of times.

Test Set 1

In Test Set 1, the number of questions $\mathbf{Q}$ is small enough that we can simply enumerate all possible $2^{\mathbf{Q}}$ sequences of answers, and filter out those who are inconsistent with the input (i.e., those for which one of the students would obtain a different score than they actually got). From the consistent ones, we can estimate the p_qs above as the ratio between the number of sequences that answer `T` to question q, over the total. Then, we can simply choose to answer `T` to those questions with $p_q > \frac{1}{2}$ and `F` to those with $p_q < \frac{1}{2}$. We can answer the questions with $p_q = \frac{1}{2}$ either way.

Test Set 2

In Test Set 2 $\mathbf{Q}$ is large, so we cannot enumerate the sequences of answers. We can, on the other hand, figure out the probabilities p_q in a different way, and then proceed as before with choosing the answers by comparing those values to $\frac{1}{2}$.

An insight-based solution

One way to solve Test Set 2 is by splitting the questions into types. If two questions q_1 and q_2 received the same answer from each student, then by symmetry $p_{q_1} = p_{q_2}$. Then, let p_{ab} be equal to the probability of a question's answer being $\texttt{T}$ given that the first student answered a and the second student answered b to it. By the first observation, each p_q is equal to one of the 4 values $p_{\texttt{TT}}$, $p_{\texttt{TF}}$, $p_{\texttt{FT}}$ or $p_{\texttt{FF}}$. Moreover, by the symmetry of complementing answers, $p_{\texttt{TT}} = 1 - p_{\texttt{FF}}$ and $p_{\texttt{TF}} = 1 - p_{\texttt{FT}}$. Therefore, we can express every p_q as a linear function of up to two variables $p_{\texttt{TT}}$ and $p_{\texttt{TF}}$. That means we can express the expected score of both students as linear functions on those two variables too. Given that their expected score should match their actual score, that gives us two equations with two unknowns. We can derive the real values of $p_{\texttt{TT}}$ and $p_{\texttt{TF}}$ from that system of equations. With every p_q calculated, we can simply choose, for each question, an answer that has maximum probability, as in the solution for Test Set 1.

Notice that there are 4 possible cases depending on how our two variables compare with $\frac{1}{2}$ (if one is exactly $\frac{1}{2}$ that means two cases are equivalent and if both are $\frac{1}{2}$ then all cases are equivalent). The 4 cases exactly match with either answering the same as one of the students, or answering the opposite of one of the students. We can use this observation to greatly simplify the implementation as the score of a sequence given by a student is given to us, and the score of the complement of the answers of student i is $\mathbf{Q} - \mathbf{S_i}$, so we can easily obtain the scores of the 4 options and pick a highest one.

A more competitive-programming-standard solution

In case of having 2 students, we can use dynamic programming to calculate the probability of each question having a particular answer. We compute the recursive function $f(s_1, s_2, q)$ defined as "how many ways are there to answer questions $q, q+1, q+2, \ldots, \mathbf{Q}$ such that student 1 gets exactly s_1 of them right and student 2 gets exactly s_2 of them right?" We can define that function recursively as

$$f(s_1, s_2, q) = f(s_1 - I_1(q, \texttt{T}), s_2 - I_2(q, \texttt{T}), q+1) + f(s_1 - I_1(q, \texttt{F}), s_2 - I_2(q, \texttt{F}), q+1)$$

where $I_i(q, c)$ is 1 if student i answered c to question q and 0 otherwise. The base cases are $f(0, 0, \mathbf{Q}+1) = 1$ and $f(s_1, s_2, \mathbf{Q}+1) = 0$ whenever one of s_1 or s_2 is not 0. To simplify memoization, we may want to add a case to simply answer 0 whenever s_1 or s_2 are negative, but the recursive equation holds as written for those cases.

By memoizing that function, we can compute it in time $O(\mathbf{Q}^3)$. We can calculate the probability of question 1's answer being $\texttt{T}$ as

$$\frac{f(\mathbf{S_1} - I_1(1, \texttt{T}), \mathbf{S_2} - I_2(1, \texttt{T}), 2)}{f(\mathbf{S_1}, \mathbf{S_2}, 1)}.$$

Then, by symmetry, we can reorder the questions to make any question the first one and re-run to compute the probability for any question. After having all probabilities, we simply answer the most likely answer for each question and sum its probability to our expected score. Since we need to run the probability computation $O(\mathbf{Q})$ times (once per question), the overall algorithm takes $O(\mathbf{Q}^4)$ time. This can be a little too slow.

If we add only the first observation of the insight-based solution, we can notice that two questions that were answered the same by both students have identical probabilities. Then, we only need to calculate the probability of up to 4 question types (in the notation of the previous solution, we use dynamic programming to calculate all the p_{ab}s). This improves the overall running time to $O(\mathbf{Q}^3)$, which fits better within the time limit. An observation about complement

could further reduce this to only 2 question types, but that does not change the time complexity and it is not needed to pass this test set.

Test Set 3

The dynamic programming solution for Test Set 2 can be generalized to Test Set 3 by adding an additional score as another parameter to the recursive function. However, the additional dimension and larger limit for $\mathbf{Q}$ can easily make such solutions too slow.

Combining the full insights of the first solution to Test Set 2 with the solution to Test Set 1 works, though: there are 8 probability variables p_{abc}, and pairs of complementary variables have complementary probabilities, so we only care about 4 different ones.

Let us call the subindex of the variables (the abc part) the "type" of a question. Let us number the types 1 through 4 in any order. If there are q_j questions of type j, we can use quadruples (t_1, t_2, t_3, t_4) with $0 \le t_j \le q_j$ for all j to represent sequences of answers that answer `T` to exactly t_j questions of type j. We know that there are

$$\binom{q_1}{t_1} \cdot \binom{q_2}{t_2} \cdot \binom{q_3}{t_3} \cdot \binom{q_4}{t_4}$$

sequences of answers represented by this particular quadruple. If we filter the quadruples by the ones that give each student their actual score, we are effectively enumerating answers that are consistent with the input. This is what we did for Test Set 1! In this way, we can count which amount t_j of questions of type j has the largest probability and choose that one, for each j.

There are at most $(\mathbf{Q}/4)^4$ quadruples to check. This makes the time complexity of the algorithm $O(\mathbf{Q}^4)$, but the $1/256$ constant is pretty significant, and a good implementation runs comfortably in time.

But wait! We can refine this solution even more by using the solution for Test Set 2 that ends with solving the system of equations! We can express the score of each student as a linear function of t_1, t_2, t_3 and t_4. That gives us a system of 3 equations and 4 unknowns. That means that we only need to try all possible values for one of the t_j and then simply solve the system to find unique values for the other three. That refines the solution above to requiring only $O(\mathbf{Q})$ time.