Kick Start 2017 - Round C

Analysis: Magical Thinking v2

Magical Thinking: Analysis

Small dataset

In the Small dataset, there is only one other friend to consider, and we know how many questions $\mathbf{S_0}$ they got right. We need to choose exactly $\mathbf{S_0}$ of the questions to be the ones they got right (call this set R); that means they got the other $\mathbf{Q}$ - $\mathbf{S_0}$ questions wrong (call this set W). Moreover, we need to do this in the way that maximizes our own score.

Let's divide the questions into two types: type A when we gave the same answer as our friend, or type D when we gave different answers. Suppose that we assign the questions to sets R and W in some way, and suppose that there is a type D question in set R, and a type A question in set W. Then we got both of those questions wrong! But if we swap the two, then we instead got both questions right. So it is always advantageous to make such swaps, and we should always load up set R with as many type A questions as we possibly can. If there are not enough type A questions to do this, then we must fill up the rest of set R with type D questions.

We can either implement this greedy strategy, or generalize it into a formula: if X is the number of questions on which we agree with our friend, then the answer turns out to be $\mathbf{Q} - |\mathbf{S_0} - X|$.

Another possible approach for the Small dataset is to use brute force. Even in the worst case of $\mathbf{Q} = 10$, there are only $2^{10} = 1024$ different possible sets of answers. We can check all of them and see which ones are consistent with our friend's score, and which of those gives us the highest score.

Large dataset

The Large dataset introduces two additional complications. For one thing, there can now be up to 50 questions, so the brute force approach mentioned above won't work; 2^{50} is over 10^{15}, and there are far too many possible sets of answers to check.

Our greedy method above will work for $\mathbf{N} = 1$, $\mathbf{Q} = 50$, but adding a second friend introduces some complications. For example, trying to make greedy decisions about one friend in the same way we did for Small dataset 1 might cause the other friend to end up with the wrong score. However, we don't need a greedy approach, because we can take advantage of the small number of types of questions. When $\mathbf{N} = 2$, there are only four types:

1. Both friends agree with us.
2. Only friend 1 agrees with us.
3. Only friend 2 agrees with us.
4. Neither friend agrees with us.

We can try every possible tuple (a, b, c, d) of the numbers of Type 1, 2, 3, and 4 questions that we got right, with 0 ≤ a ≤ the number of Type 1 questions, 0 ≤ b ≤ the number of Type 2 questions, and so on. Any tuples that do not cause both friends to have the correct scores should be discarded. We can then choose the remaining tuple for which a + b + c + d is maximized. This method is $O(\mathbf{Q}^4)$, but the greatest possible number of such tuples to check occurs when $\mathbf{Q} = 50$ and there are 12 or 13 of each of the four types, and even then, there only about 25000 possibilities to check.

Another option for this dataset is to use dynamic programming. For example, we can start with a set containing only the list [0, 0, 0], which represents that at the start of the test, before any questions are answered, we have 0 points, friend 1 has 0 points, and friend 2 has 0 points. Then, suppose that our answer to the first question is F, our first friend answered T, and our second friend answered F. There are two options: either the real answer to the first question is F (for which case we create the list [1, 0, 1], since we and the second friend have earned a point, but the first friend has not), or the real answer is T (for which case we create the list [0, 1, 0]). Then we replace our old set with the new set ([1, 0, 1], [0, 1, 0]), and for each list in that set, we in turn consider what happens when we answer F or T for the second question, and so on. Once we have finished, we check for the list in which our friends' scores are both correct and we have the highest possible score.

Why is this method any better than brute force? The key difference is that if we create a list that is already in our set, we do not add another copy of it, so we do not waste time individually considering many nearly equivalent (for our purposes) answer sets. Because our scores and our friends' scores can range from 0 to **Q**, inclusive, there are $(\mathbf{Q} + 1)^{N + 1}$ possible lists; since (as an upper bound) we have to check all of them once for each question, the method is $O(\mathbf{Q}^{N + 2})$. This is $O(\mathbf{Q}^4)$ when **N** = 2. In practice, this is a "slower $O(\mathbf{Q}^4)$" than the method above, since we may need to check $51^3 \times 50$ = over 6×10^6 possibilities, but it is still easily fast enough to solve the Large dataset. The approach also works just fine with a three-dimensional array instead of a set, but the set lets us avoid checking for the existence of all possible lists at each step of the process.