Code Jam 2020 - Qualification Round

Analysis: ESAb ATAd

Test Set 1

In Test Set 1, there are only 10 positions in the string. We can query for each of them and then submit the complete string, without having to worry about any quantum fluctuations (which would only happen if we submitted an 11th query).

Test Set 2

Here is one of various ways to solve the second test set. We begin by querying for the first ten positions in the real string, then create a "possibility set" containing all 1024 20-character strings that begin with those 10 characters. Then we update our "possibility set" to contain all strings that could have arisen from those strings after the next quantum fluctuation. The correct answer is in here somewhere — now we need to narrow the set down!

Before making each subsequent query, we first find the string index (between 1 and 20) at which the proportion of `0`s and `1`s among the strings in our possibility set is most nearly even. Then we query the real string at that index, and eliminate from the possibility set any strings that are not consistent with that information. Whenever we can indeed find a position with even proportions, we are guaranteed to cut the size of the set in half, but if there is no such position, we may not be able to eliminate that many possibilities. We can continue in this way, remembering to expand the possibility set every time there is a quantum fluctuation, until only one possibility remains, which must be the answer.

It is not easy to prove that this strategy will converge upon an answer. Intuitively, we can observe that a quantum fluctuation increases the size of the possibility set by at most 4, and even if we somehow only cut the possiblity set by 20% with each pruning, we would still easily beat that factor-of-4 increase and make enough progress to finish within 150 queries. Moreover, it would not be possible for the strings in the possibility set to all be distinct while being so similar at *every* individual position (recall that we always pick the position that will be *most* useful to us in the worst case). Also, Test Set 2 is a Visible Verdict set, so we might as well just submit our answer and see.

Test Set 3

The above strategy will not work for 100-character strings, since the possibility set would be astronomically huge. Fortunately, there is a much simpler approach.

Observe that if we can find two positions that are equidistant from the center of the string and have the same value, we can use them to detect when a quantum fluctuation has included a complementation (with or without a reversal). Suppose, for example, that the two ends of the string are `0` just before a quantum fluctuation. After the fluctuation, we can check the first one. If it is `1`, then there was a complementation; if not, there wasn't one. This is true regardless of whether that quantum fluctuation included a reversal.

Now suppose that we continue to check pairs of positions in this way, moving inward one step at a time. After every quantum fluctuation, we must spend one query to check for complementation so we can update our existing knowledge about the string if there has been one. If every pair turns out to be a "same pair" like the first pair, then we never needed to care about reversals anyway (since the string is palindromic), and we are done.

But what if, in the course of this, we find a "different pair"? Such pairs are helpful in their own way! If we query the first position of a "different pair" after a quantum fluctuation and we find that that bit has changed, then we know that either a complementation or reversal has happened, but not both.

Once we have such a "different pair", we can use it in conjunction with the "same pair", spending 2 out of every 10 queries to learn exactly what happened in each quantum fluctuation. For example, if the first position of our "same pair" stayed the same but the first position of our "different pair" did not, we know that the quantum fluctuation included a reversal but no complementation.

In the above analysis, we assumed we would encounter a "same pair" first. If the first pair is different, though, we can proceed until we encounter a "same pair"; if we never encounter one, then we do not care about the distinction between complementation and reversal, because the operations are equivalent for that particular string. If we do encounter a "same pair", though, then we can proceed as above.

How many queries will we need in the worst case? We can use all of our first 10 to gather data, since whatever happened in the quantum fluctuation at the start of the problem is unknowable and does not matter. After that, we may need to use up to 2 out of every 10 queries to reorient ourselves before spending the remaining 8 gathering data. So, to be sure we can find the entire string, we will need 10 queries, plus 11 more sets of 10 queries in which we learn 8 positions each time, (to get us to 98 positions known), plus 2 more queries for a final reorientation, plus 2 more to get the last two positions. That is a total of 124, which is well within the allowed limit of 150.

Regarding the name...

Last year, we had the Dat Bae problem about deletions from a string in a database; the name was `Data Base`, altered in a way that reflected the theme. `ESAb ATAd` is similar, with case change serving as a rough equivalent of complementation. (Imagine how much the Code Jam team has enjoyed trying to type the name correctly each time!)