Code Jam 2022 - Round 1B

Analysis: ASeDatAb

Test Set 1

Whatever we try to do, the randomly-rotating judge in this test set might scuttle our plans. But we can fight randomness with randomness!

First, we can observe that if the judge ever tells us that the record has eight 1 bits, we have all but won (as long as we have at least one interaction left). This is because we can then submit 11111111, and no matter how the judge rotates it, the final result of XORing will be 00000000. Because of this, if we are ever told that the record has more than four 1 s, it should be to our advantage to aim for eight 1s rather than zero 1s!

For now let's suppose the judge tells us there are b bits in the record, for some b between 1 and 4. (We will handle the other cases by symmetry, as explained above.) Let's do the following: choose a string with b bits uniformly at random, then send that. Now it doesn't really matter how the judge rotates our string -- whatever the resulting rotated string is, we were just as likely to pick that in the first place.

What are the possible outcomes? Suppose that $b = 2$. Then we have a $\frac{1}{\binom{8}{2}} = \frac{1}{28}$ chance of flipping both of the two 1s (and therefore winning!), a $\frac{\binom{2}{1}\binom{6}{1}}{\binom{8}{2}} = \frac{12}{28}$ chance of flipping one of the two 1s and some innocent 0 (and thus being back in the same $b = 2$ boat), and a $\frac{\binom{2}{0}\binom{6}{2}}{\binom{8}{2}} = \frac{15}{28}$ chance of missing both 1s and creating two new 1s (taking us to $b = 4$). This may not seem too promising so far, but hang on...

Doing the same sort of analysis for the state $b = 4$, we find that we end up at one of $b = 0, 2, 4, 6, 8$, with probabilities $\frac{1}{70}, \frac{16}{70}, \frac{36}{70}, \frac{16}{70}, \frac{1}{70}$, respectively. But, as we mentioned, being at $b = 6$ is just the same as being at $b = 2$. If we are at $b = 6$, we can try to use two 1s to flip the two 0s, in the hopes of reaching 11111111. Similarly, being at $b = 8$ is (essentially) as good as being at $b = 0$. So we will lump those two probabilities into $b = 2$ and $b = 0$, respectively, getting transition probabilities to $b = 0, 2, 4$ of $\frac{1}{35}, \frac{16}{35}$, and $\frac{18}{35}$, respectively.

Notice that if b is even, we are trapped in the even-b-verse, randomly walking (according to those transition probabilities) until we either reach $b = 0$ or reach $b = 8$ or we run out of guesses. And if we are not in the even-b-verse, one round of sending exactly $min(b, 8 - b)$ randomly placed 1s will get us there; we leave this as an exercise. So our strategy can be to spend up to two rounds getting into the even-b-verse, up to 297 rounds wandering, and up to one round possibly turning a 11111111 into a 00000000.

What are the chances that we will succeed in those 297 rounds? Observe that until we reach a winning state (00000000 or 11111111), we are always in one of two other states: $b = 2$ (lumped in with $b = 6$), or $b = 4$. In the former case, we have a $\frac{1}{28}$ chance of transitioning to a winning state, and in the latter case, that chance is $\frac{1}{35}$. Just for ease of argument, let's pessimistically guess that we get stuck hanging around in the less advantageous $b = 4$ state. But then to not win, we would still have to fail our $\frac{1}{35}$ lottery 297 times. The probability of that is $(1 - \frac{1}{35})^{297} \approx 0.0002$. So we have at least a 99.98% chance of succeeding with this strategy, and that is an overly conservative lower bound!

(If you're curious about the actual success probability, we can conservatively assume that we always start our journey at $b = 4$, and then find the upper left cell of $\begin{pmatrix} 1 & \frac{1}{28} & \frac{1}{35} \\ 0 & \frac{12}{28} & \frac{16}{35} \\ 0 & \frac{15}{28} & \frac{18}{35} \end{pmatrix}^{297} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$, which turns out to be about $0.99993 \approx 99.993\%$).

Of course, we still have to pass *all 100* test cases in Test Set 1, and the probability of this is a bit smaller: $\approx 0.99993^{100} \approx 0.993$. But 99.3% isn't so bad, and if we see an unlucky failure with this method, we can easily get another independent try by changing our code's random seed, since we have control over that source of randomness.

Test Set 2

In Test Set 2, the judge does not behave randomly and can and will choose rotation values that keep us away from reaching our goal. So, we need a strategy that is guaranteed to reset the record to all zeroes.

One way to do this is to consider the current *state*. Let's define a state as the set of all possible values the record could currently be set to. A key observation here is that two values that are cyclic rotations of each other are equivalent.

Therefore, we can eliminate these duplicates from our sets. After the first exchange, we know how many bits are set to 1 in the record. Thus, all potential states only have values that have the same bit count.

We can enumerate all possible values for each bit count (while removing duplicates that are cyclic rotations of another value). If we do this, we find the following sets of potential values:

- 0 bits: $\{00000000\}$
- 1 bit: $\{00000001\}$
- 2 bits: $\{00000011, 00000101, 00001001, 00010001\}$
- 3 bits: $\{00000111, 00001011, 00001101, 00010011, 00010101, 00011001, 00100101\}$
- 4 bits: $\{00001111, 00010111, 00011011, 00011101, 00100111, 00101011, 00101101, 00110011, 00110101, 01010101\}$
- 5 bits: $\{00011111, 00101111, 00110111, 00111011, 00111101, 01010111, 01011011\}$
- 6 bits: $\{00111111, 01011111, 01101111, 01110111\}$
- 7 bits: $\{01111111\}$
- 8 bits: $\{11111111\}$

Notice that the largest of these sets (4 bits) has only 10 elements. Thus, there are at most $2^{10} = 1024$ unique states for when we have 4 bits on. This is small enough to consider all possible states and so something similar to a BFS (breadth-first search) from the state with all zeroes.

Specifically, we can consider the set of all states that we know can be forced to reach all zeroes (initially just the solved state where the record is `00000000`). Then, for a given state, A, we can consider trying all possible values for V and simulate the results of the 8 different rotation values the judge could choose. Grouping those by their bitcounts gives us the possible states that A could transition to for a specific value of V. If all of those states are ones we have processed, then we know that when state A, we can use this value of V to get us closer to setting the record to all zeroes.

It turns out that if we keep repeating the above process, we will eventually process all possible states. This gives us instructions on which numbers to provide for V given the current state. The alternative solution is proof provided below works to prove that this is always possible for 8-bit records.

Alternative Solution

It turns out that we can solve this problem without ever knowing the bit count after interactions (other than being told when we eventually reach `00000000`).

Let's consider how we would solve this problem for a record that has only 1 bit. Since we know the value starts as not 0, we can force the state to reach 0 by sending 1 to the judge. Let's call this sequence $P[0]$:

```
1
```

Now, let's consider a record that is 2 bits (and is not all zeroes). Let's start by assuming that the two bits are the same. If that's the case, we can force the record to be all zeroes by sending 11. If We haven't reached all zeroes after that, that means our initial assumption that the left and right bit were the same was incorrect. So, if we send 10, we can make the two bits the same. Then, if we are still not all zeroes, we can send another 11. Let's call this sequence $P[1]$:

```
11    // P[0] + P[0]
10    // P[0] + 0
11    // P[0] + P[0]
```

Now, let's generalize this and assume that the record has 2^k bits and is not all zeroes. Let's assume that the left 2^{k-1} bits and the right 2^{k-1} bits are the same. If that's the case, we can use $P[k-1]$ (but each step is appended to itself) to force the record to reach all zeroes. If we did not reach all zeroes then our assumption that the left half and right were the same was not correct.

So, we can use the first instruction in $P[k-1]$ and append 2^{k-1} `0`'s to it. Then we can repeat all of $P[k-1]$ (each step doubled like before) again. As long as we keep not reaching all zeroes, we repeat this process with the next instruction in $P[k-1]$.

The following Python code shows this process for how we can generate $P[3]$ for 8-bit records:

```
def appendzero(s):
  return s + '0' * len(s)

def expand(s):
  return s + s

def P(k):
  if k == 0:
      return ['1']
  seq = P(k - 1)
  seq_with_zero = [appendzero(s) for s in seq]
  seq_with_copy = [expand(s) for s in seq]
  res = seq_with_copy[:]
  for ins in seq_with_zero:
      res += [ins]
      res += seq_with_copy
  return res

print(P(3))
```