Code Jam 2019 - Round 3

Analysis: Zillionim

There may not be a *zillion* possible solutions to this problem, but there are more than we can discuss in this analysis! We'll only touch on a few here, some of which work, and some of which don't. You're welcome to share yours in the Code Jam Google Group!

Random play

What if we always choose a move uniformly at random from among all available moves, just like the AI does? Intuitively, since the overall number of coins is large relative to the size of any one move, going first or second is not very important (given that in this case, both sides are playing randomly and not using any strategy), and every game is close to a fair coin flip. Suppose that we have a 50% chance of winning a game with this strategy; then, per the binomial distribution, we have only about an 0.0004% chance of winning 300 or more games out of 500. So this approach won't even get us the 1 point for test set 1!

A mirror strategy

If we were allowed to go first in this game, we could guarantee a win using the following strategy. Imagine a "center line" drawn between the (10^{12} / 2)-th and ((10^{12} / 2) + 1)-th coins, dividing the coins into two regions of equal size. On the first turn, we could take the group of 10^{10} coins bisected by this line — that is, the coins numbered from ((10^{12} - 10^{10}) / 2) + 1 to ((10^{12} + 10^{10}) / 2). Then, after every move by the opponent, we could make the same move, but reflected across this center line. For example, if the opponent were to take the 10^{10} coins starting from coin number 2, we would respond by taking the group of 10^{10} coins starting from the next-to-last coin and going left. We would always be guaranteed a move, by symmetry, so the opponent would have to eventually lose.

Sadly, we cannot move first, but we can try to adapt this strategy by just making the mirrored version of the opponent's move. This will always be possible unless the opponent makes a move that crosses the center line, in which case we cannot mirror it. But if the opponent happens to move close to the center line without crossing it — specifically, taking a coin fewer than 10^{10} / 2 coins away from the center without crossing the center — then they will ensure their own defeat. If they do happen to make a move that crosses the center line, we can abandon our strategy and move randomly.

We can run a local simulation and find that this strategy does better than randomness, winning around 57.5% of the time. Unfortunately, this only gives us about a 14% chance of passing test set 1. Moreover, because our moves depend on the judge's moves, we cannot easily tweak the judge's randomness; we only get some control over that once the judge has made a move that hurts our strategy. So, all in all, it's probably best to abandon this approach.

A 2 × 10^{10} strategy

Let's call a remaining set of (at least 10^{10}) contiguous coins a "run". Observe that a "run" of exactly 2 × 10^{10} remaining coins can be very useful for us. We have the option of taking the first 10^{10} or the last 10^{10} coins from that run, and leaving a run of exactly 10^{10} coins (and therefore one possible move) behind. On the other hand, if we take any other group of 10^{10} coins from that run, we will leave no moves behind, "destroying" the run. Also notice that if the AI happens to move within this run, it will virtually always make the second type of move; the odds of it happening to choose the first or last 10^{10} coins are negligible.

Because of this, a run of exactly 2×10^{10} coins lets us control the parity of the game. Suppose, as a thought experiment, that the only remaining runs are of exactly 2×10^{10} coins each, and that it is our turn. If the number of remaining runs is odd, we can move in a way that destroys one of the runs, and then the AI will do the same, and so on until we leave the AI with no moves. If the number of remaining runs is even, we can take the first 10^{10} coins from one run, leaving the other 10^{10} behind as a smaller run. Now the AI is in the same bad "even number of remaining runs" situation that we were just in.

We can set up a situation like this by making many moves early on that leave behind runs of exactly 2×10^{10} coins. For example, we can repeatedly choose the largest remaining run of size 3×10^{10} or greater, and start from the $(2 \times 10^{10} + 1)$-th coin from the left in that group, leaving a run of exactly size 2×10^{10} behind (in addition to any leftover piece to the right of our move). Then, as long as there are still runs larger than 2×10^{10} but smaller than 3×10^{10}, we can use our moves to destroy them by moving in their centers. Our goal is to eliminate all runs larger than 2×10^{10} before the AI randomly destroys all of our runs of 2×10^{10}. Intuitively, we do not need to worry too much about this, since the AI is more likely to move within some remaining larger runs than within our last remaining run of 2×10^{10} runs, so it will usually be helping us out! Once all runs are of size 2×10^{10} or smaller, and we have at least one run of size 2×10^{10}, we have imposed the near-lockdown situation described above.

We hope you will forgive us for not trying to calculate an exact success probability for the above strategy, but one can use a local simulation to show that it wins, e.g., all 100000 of 100000 games. The chances of it losing more than one of 500 games are vanishingly small... and even if the worst somehow happens, in this case, we do have some control over the overall randomness, and we can try again with a slight tweak.

Other parity-based strategies

In Round 1C this year, we had Bacterial Tactics , another problem about a turn-taking game. You may recall that the analysis brought up the Spragueâ€“Grundy theorem ; can we use a similar parity-based strategy here?

As in our 2×10^{10} strategy, we can try to keep the number of remaining runs even for the AI by moving in the middle of a run if there is an even number of runs, or otherwise taking the left end of some other large run. Empirically, this strategy solves TS1, and it even solves TS2 if we always take the largest remaining run. It ends up being similar to our best strategy described above, even though it does not allow for such fine control.

We can even achieve a perfect solution for this problem by exhaustively calculating Grundy numbers and storing them using run-length encoding! However, it is possible to arrive at the 2×10^{10} idea above, which is good enough, without knowing anything about this theory.