Code Jam to I/O for Women 2019 - Code Jam to I/O for Women

Analysis: War of the Words

It might seem impossible to promise that we will achieve any particular score in a game in which we do not know which options are better than others, and we don't even know when we are scoring points! But we have our ways...

Test set 1

At first, we might be tempted to send a totally random word every turn. Unfortunately, that's not enough to pass the first test set! When we pick a random word W, the robot will respond with a word R chosen uniformly at random from all words ranked higher than W. Since our W will have a 50th percentile rank on average, the robot's R will have a 75th percentile rank on average, so we might expect to score around 25 points in each case (less if we tie). But we have to get at least 25 points in each of 50 test cases. Good luck winning 50 coin flips in a row!

One approach against an unknown strong opponent, familiar to practitioners of aikido, is to use the opponent's own strength against them:

- Play some fixed but arbitrary word W — say, `HIGCJ`. The robot responds with some word R_1.
- Play W again. The robot responds with some word R_2.
- Now "borrow" and play R_1; the robot responds with some word R_3.
- Play W again. The robot responds with some word R_4.
- Now "borrow" and play R_3; the robot responds with some word R_5...

Suppose that our initial choice of W ranks somewhere around the middle of the pack. We certainly lose by playing W against R_1, since we know that R_1 beats W. But then when we borrow R_1 to play against R_2, we know that both R_1 and R_2 were chosen uniformly and independently at random from among words that beat W, so we should a slightly less than 50% chance of winning (because R_1 and R_2 might be the same, and we do not score points for ties). Then we lose again by playing against R_3, but when we borrow R_3 to play against R_4, we are playing a word that beat a word that beat W against a word that beat W, so we should have around a 75% chance of winning. As this strategy unfolds, we should lose every other round, but win the remaining rounds with probability 50%, 75%, 87.5%, and so on. This seems like it should easily be enough to get us to 25 points.

However, there is a flaw in this argument: as our "borrowed" words get stronger and stronger, at some point, one of them may be the highest-ranking word, in which case the next "borrowed" robot word will be the lowest-ranking word. Then our "borrowed" words, which are supposed to get us our wins, are actually weak and will cause us to lose those rounds. But now we start to win on the rounds on which we play W, since the robot is playing weakish words in response to our weak "borrowed" words, and W probably beats them. So we still get our wins, but now from the W rounds, until the cycle passes by W again. Notice that this implies that the rank of our initial choice of W was not important.

It turns out that this strategy scores close to 50 points on average, but with enough variance that it will not pass test set 2. (We do not need to worry about it failing test set 1, though. In general, running simulations can help us get estimates of the variance of random solutions like this.)

Test set 2

The most notable characteristic of the game is the fact that the otherwise lowest-ranking word L beats the highest-ranking word H. Can we exploit this irregularity? Suppose that we had a way of knowing the identities of L and H. Then we could score a point on *every* turn by repeating the following cycle:

- We play H, forcing the robot to play L in response.
- We play some arbitrary word that is not H or L, beating L and earning a point. The robot plays some other word W in response.
- We beat W with H, forcing the robot to play L in response, and so on. (If W just happened to be H, which is unlikely, then we play L in response instead, and the robot plays another word W' in response, and so on.)

So, if we can confidently find L and H in fewer than 50 turns, we can certainly pass.

One strategy is as follows. Start with an arbitrary word, and then spend a number of turns copying the robot's most recent response. Since the robot always has to play something higher-ranking in response, and our play will always be of the same rank as the robot's last word, this algorithm will gradually visit higher-ranked words. Eventually, there must be an exchange in which we play H and the robot is forced to play L. Then we climb the ranks again from there. So, this strategy will repeatedly "cycle" us through the rankings. Although we will not necessarily visit the same particular subset of ranks each time, each cycle will contain H followed by L.

How many of our turns (not counting the robot's turns) will it take us, on average, to reach the H to L transition? It will take 0 turns if we are lucky enough to start there, 1 turn if we start 1 rank away, an expected 1 + ((0 + 1) / 2) = 1.5 turns if we start 2 ranks away, an expected 1 + ((0 + 1 + 1.5) / 3) = 1.8333... turns if we start 3 ranks away, and so on. These are the harmonic numbers. If we start with a random one of our 10^5 words, the mean number of turns to reach the highest-ranking word is about 11.09. So if we spend 50 turns, we can go through at least three cycles, with high probability.

Then, our program can look through the results to identify these H to L transitions; we should see multiple instances of those two words back to back. There may be a few other instances of consecutive ranks that appear in multiple cycles — for example, maybe the robot always happens to choose the second-highest-ranking word on its way to the highest-ranking word. But the H to L must be present in every cycle, and even if we are unlucky and other consecutive patterns also appear in all of our cycles, we can still reason that the one that occurs closely after all the others is the real transition, since other consecutive patterns are only likely to occur closely before the transition.

Another approach is based on the fact that there are only two words that leave the robot with a single choice: if we play the second-highest-ranking word, the robot must play the highest-ranking word, and if we play the highest-ranking word, the robot must play the lowest-ranking word. So, when the robot sends us a word W, we can send two instances of W and note the robot's two responses. If they are different, we pick the first one and send two instances of that, and so on. If they are the same, we can send W a few more times to see whether we always get the same response R. Then we can send R a few times to see whether we always get the same response S (in which case R is the second-highest-ranking word and S is the highest-ranking word) or varying responses (in which case R is the highest-ranking word and S is the lowest-ranking word).

You can experiment locally and run your own simulations until you are confident that you have a solution that will pass the test sets with high probability. Since both test sets are Visible, and the judge is deterministic, it is possible to tweak your solution and try again if you get unlucky with a reasonable approach. Any overall success probability over, say, 50% should suffice, but the approaches described above should do much better than that.