Code Jam to I/O for Women 2018 - Code Jam to I/O for Women

Analysis: Tricky Trios

Tricky Trios: Analysis

Playing the game

Before we can approach either of the datasets, we need to figure out an optimal strategy for the game. One important insight is that we should not begin a round by flipping over a known card, unless we know where all three cards in a trio are (in which case we can flip them all to remove that trio).

To see why this is, suppose that we begin by flipping the one or two known cards with a certain number. Then, when we move on to flipping unknown cards, we either find the one or two remaining cards we need, or we fail and learn the identity of only one more card. But if we instead begin by flipping an unknown card, we have exactly the same chance of encountering the cards needed to remove the aforementioned trio, in which case we can end the round by flipping the known ones. We also have some other advantages: we have a chance of removing any of the other trios, and if we fail to do that, we will learn the identities of two more cards. So, beginning with an unknown card is strictly better than beginning with a known card; it leaves more options open and gets us more valuable information.

What about the parts of a round other than the beginning? There is one situation in which we have some flexibility: if the unknown card revealed on our first flip matches one known card, there is no harm in flipping over that known card as our second action before flipping over an unknown card as our third action. But it is no worse to flip over an unknown card as our second action instead, since we need to find the last member of that trio either way.

So, the following simple rule is optimal: only flip known cards when they are the last ones needed to remove a trio.

Small dataset

This is an unusual dataset for Code Jam: there are only five possible test cases, and three of them are given away as samples, so we can hardcode those in and only focus on solving the **N** = 3 and **N** = 4 cases.

Notice that whenever we flip an unknown card, we have no basis for choosing any particular unknown card. Without loss of generality, we can choose to reveal them from left to right. With this in mind, one tempting approach is to repeatedly simulate playing the game, each time choosing a random left-to-right order in which to deal with the cards, and then take the average number of rounds. The tight requirement of an absolute or relative error of 10^{-6} makes this challenging, though; even millions of simulations might not get us close enough! We can let a simulation run for quite some time and get answers before starting the 4-minute submission window, but even optimized code may not be fast enough, especially in an interpreted language, for example.

Instead of simulating random left-to-right orders, can we enumerate them all, play through each one, and then take the average number of rounds? For our **N** = 4 case, there are a total of (12 choose 3) × (9 choose 3) × (6 choose 3) × (3 choose 3) = 369600 orders, which is a tractably small amount. Although it is not necessary, we can cut this down by another factor of **N**! by noting that the numbers on the cards are essentially interchangeable; an order like

111222333444 will yield the same result as an order like 222333444111. This drops the number of cases to 15400 for **N** = 4, and 1401400 for **N** = 5. Beyond that point, though, there are just too many orders to consider individually.

Large dataset

At the start of each round of the game, for each trio that we have not already removed, we know the locations of three, two, one, or zero of the cards, depending on what we have flipped over on earlier rounds. Let us classify trios accordingly as Three-Known, Two-Known, One-Known, or Zero-Known. Then we can consider the number of trios of each type — (K_3, K_2, K_1, K_0) — as a *state*, and think of a round as a transition from one state to another state. We begin the game at (0, 0, 0, **N**), and we want to get to (0, 0, 0, 0) as efficiently as possible, but our path will depend on how lucky we get when flipping over unknown cards.

Let us use f(state) to denote the question "how many rounds will it take, on average, to finish the game when starting from this state?" To calculate f(state), we must consider all the states in which the round can end, and the probabilities of reaching each of those destination states. Then f(state) is a sum of the f()s for those destination states, weighted by the probability of reaching each of them.

We can simplify this model at the outset by noting that if we play according to our strategy above, K_3 will never be larger than 1, and if it is 1, we should spend the next round to immediately remove the Three-Known trio. That is, $f(K_3, K_2, K_1, K_0) = 1 + f(K_3 - 1, K_2, K_1, K_0)$. So we can remove the K_3 term and work with states of the form (K_2, K_1, K_0), and add in an extra round as needed to reflect removing a Three-Known trio.

For example, suppose that we start a round in a state (1, 1, 1); this means that there are six unknown cards remaining, with one from a Two-Known trio, two from a One-Known trio, and three from a Zero-Known trio. (Remember that there are other known cards around as well — for instance, the other two cards from the Two-Known trio — but we are focusing on the unknown cards.) Let us consider one possible scenario: our first draw turns out to be from the One-Known trio, and then our second draw turns out to be from the Two-Known trio, which ends the round. This converts our Two-Known trio into a Three-Known trio and our One-Known trio into a Two-Known trio. We should immediately spend the next round to remove the Three-Known trio.

The probability of the above happening is 2/6 (the odds of drawing one of the unknown cards from the One-Known trio) × 1/5 (the conditional odds of drawing the unknown card from the Two-Known trio). So, when we write the expression for f(1, 1, 1), it should include the term 2/6 × 1/5 × (2 + f(1, 0, 1)). The 2 represents the current round plus the extra round spent removing the Three-Known trio. Note that (1, 0, 1) reflects that we have converted a One-Known trio into a Two-Known trio and lost another Two-Known trio.

The above illustrates the calculation of just one term, and our solution needs to be able to handle any state; the more general version of the term above would be $((2 \times K_1) / (K_2 + 2 \times K_1 + 3 \times K_0)) \times (K_2 / (K_2 + 2 \times K_1 - 1 + 3 \times K_0)) \times (2 + f(K_2, K_1 - 1, K_0))$. Setting up terms like this is the heart of the problem, and there are various possible pitfalls, e.g.:

- If our first flip on a round reveals a card from a Zero-Known trio, our second flip might reveal a card from any of the following: a Two-Known trio, a One-Known trio, the same Zero-Known trio, or a different Zero-Known trio. It is important to distinguish between the latter two possibilities.
- We must avoid considering terms that represent impossible situations; we cannot draw a card from a different Zero-Known trio if there is only one Zero-Known trio in our starting state.

Once we have all this nailed down, we need a way to avoid computing f() for the same state more than once. We can store each result and then look it up again later if we need it for a future calculation, instead of doing redundant work. This is memoization, a form of dynamic programming. This optimization allows the entire Large dataset to be solved in seconds.