Code Jam 2016 - Round 1A

Analysis: Rank and File

Rank and File: Analysis

Small dataset

A natural first approach is to reconstruct the grid by brute force: try all ways of assigning the lines to the rows and columns until the lines all overlap perfectly, with no conflicts. But as we get into later rounds of Code Jam, pure brute force becomes less likely to work for every Small dataset! In this case, with **N** = 10, we can have up to 19 lines that we must turn into a 10 x 10 array. Trying all 20! = about $2.4 * 10^{18}$ ways of assigning those 19 lines (plus the one empty line that we need to figure out) to the 20 rows and columns would just take too long, and Sergeant Argus isn't going to give us that much time.

Here's one approach that takes advantage of the unusual properties of the grid. Let's continue with our 19-line case. We'll assume (without loss of generality) that the grid has a column missing. In that case, our grid has 10 rows. We can try all 19 choose 10 = 19! / (10! * 9!) = 92378 ways of choosing ten of our lines to call the "rows". But what order should they go in? Well, we know that the values in the first column, like the values in any other column, must be in strictly increasing order. If any two of the "rows" that we've chosen begin with the same number, then we must not have chosen the right set, and we can move on. Otherwise, we know exactly how to order them, so we already have a complete potential grid! Then we can just check the 10 columns to see if 9 of them match the 9 lines we didn't use in this set. If they do, then the remaining line is the answer.

Large dataset

With up to 99 lines, the approach above won't work: 99 choose 50 is roughly $5 * 10^{28}$.

It is possible to write a solution that reconstructs such a large grid. But it turns out that we don't need to frame the problem that way at all! We haven't fully taken advantage of the fact that we're only asked for the missing row or column, not for the entire grid.

In a *complete* set of lists of rows and columns of a grid, every cell of the grid appears twice: once in the list for the row it is in, and once in the list for the column it is in. There may be multiple copies of the same number in the grid, but we can still say that every number appears an even number of times in total within that complete set of lists.

But what about our input set of lists, which is missing the list for one row or column? All of the numbers in that missing list are different, and each of them appears one time fewer in the input than it would in the complete set of lists. This means that all of those numbers appear an odd number of times in our input. Moreover, they are the *only* numbers that appear an odd number of times in our input!

Therefore, we don't need to build a grid at all. All we need to do is look through all the numbers in all the lists and see which **N** of them appear an odd number of times; those are the numbers in our missing row/column. We can put them in the order they must go in (strictly increasing), and then we have our answer. And we haven't even done a single push-up!