Kick Start 2017 - Round C

Analysis: The 4M Corporation

The 4M Corporation: Analysis

Small dataset

The Limits section of the problem statement tells us that the answer to every Small test case that is not `IMPOSSIBLE` is less than 14. This suggests that brute force is a viable option for the Small dataset. The tight bounds imposed by **MINIMUM** and **MAXIMUM** help us out as well. But let's think about the worst case: what if those two values are 1 and 8, respectively, and what if the answer ends up being 13? In that case, we know that two of the values in the list must be 1 and 8, but if any of the others can be any integer in [1, 8], then we can have 8^{11} — over 8 billion — possibilities for the remaining numbers. That's too much to handle in our limited time!

Fortunately, we can cut this daunting space of possibilities down to size by avoiding redundant lists. In this problem, we don't care about the difference between [1, 1, 2, 3] and [2, 1, 3, 1], for instance. One way to avoid worrying about such differences is to only consider lists that are sorted in nondecreasing order. Even in the worst case mentioned above, there are only about 75,000 different nondecreasing lists that accord with the Small limits. (You can use a combinatorial trick to find this value, or you can just experiment and see.)

One simple way to actually generate those lists is to start with a list consisting of only **MINIMUM**, then generate all new legal lists that append one more value to the end of that, then generate all new legal lists that append one value to the ends of *those* lists, and so on. Since the number of such lists is small, an approach like this is tractable. Once you have all the lists, you can check each of them to see if it has the desired properties, and return the length of the shortest such list, or `IMPOSSIBLE` if none of the lists works.

Large dataset

With values in the [1, 10000] range, and no upper bound on the answer, we will never be able to generate all possible lists in time. We need a general, non-brute-force solution.

Let's start with some special cases that might be easy to overlook:

- There are five sets of demands that make the task clearly `IMPOSSIBLE`, because they contradict the mathematical definitions of minimum, maximum, mean, and median:
  - **MAXIMUM** < **MINIMUM**
  - **MAXIMUM** < **MEDIAN**
  - **MAXIMUM** < **MEAN**
  - **MEDIAN** < **MINIMUM**
  - **MEAN** < **MINIMUM**
- The problem might be solvable with one department. This is possible if and only if **MINIMUM**, **MAXIMUM**, **MEAN**, and **MEDIAN** are all the same, so we can easily confirm or rule out this case.
- The problem might be solvable with two departments. This requires that **MEAN** and **MEDIAN** are both exactly (**MINIMUM** + **MAXIMUM**) / 2). If this is true, we have no reason to consider using more departments, so we are done.

Otherwise, suppose that the problem is solvable with three departments. Then those three values must be **MINIMUM**, **MEDIAN**, and **MAXIMUM**. Assuming that these values do not already have the correct **MEAN**, perhaps we can insert more values until they do. We can repeatedly insert one value to the left of the median and one value to the right. First, we figure out how much the mean of the existing three values deviates from the desired mean. (To avoid errors associated with floating-point computation, it is better to compare the values' sum with **MEAN** times the number of values.) Suppose (without loss of generality) that the existing sum of values is too large. Then we can figure out which two values to insert to bring down this surplus as much as possible; usually, they will be another copy of **MINIMUM** (on the left) and another copy of **MEDIAN** (on the right). If inserting even one such pair won't reduce the surplus — that is, if **MEDIAN** + **MINIMUM** is no smaller than 2 × **MEAN** — then this method will never work. Otherwise, we can keep doing this until the surplus is gone. If inserting that last pair of **MINIMUM** and **MEDIAN** would overshoot the surplus by an amount K, we can just insert **MINIMUM** + K instead of the usual **MINIMUM**; this value will never exceed **MEDIAN**.) We don't even need to carry out each such step individually; we can figure out exactly how many will be needed.

Note that since we always insert one value to the left of our original **MEDIAN** and one to the right, the list always has a copy of **MEDIAN** in the right place. Also, given any other valid list of odd length, we can sort that list's values in nondecreasing order and pair them off (ignoring the first, last and middle elements, which are fixed): the second value with the second-to-last value, etc. Each of the pairs produced by our method contributes maximally towards reducing the surplus (except possibly the last one of value **MINIMUM** + K). So no valid list of odd length can be strictly shorter, as the total contribution would be strictly smaller.

We aren't done yet, though, because the above method only produces lists with an odd number of elements. So, regardless of what that method determined, we need to try something similar starting with four departments: **MINIMUM**, **MEDIAN**, **MEDIAN**, and **MAXIMUM**. (You may notice that we could also consider **MEDIAN** - 1 and **MEDIAN** + 1, among other possible pairs, in place of **MEDIAN** and **MEDIAN**. But there is no reason to consider this, because it would only restrict the range of new values we can insert, and thus require more values to adjust the mean.) Once we get our answer from this four-department method, we can take the smaller of its answer and the three-department method's answer, or `IMPOSSIBLE` if neither method worked.

This solution runs in constant time. Our efficiency-loving bosses at the 4M Corporation will certainly be happy with that!