Code Jam 2017 - Round 1A

Analysis: Ratatouille

Ratatouille: Analysis

Small dataset

The number of grams in a given package is not important in and of itself; what is important is which multiples of the recipe that package could be used in. If the base recipe requires 10 g of tomato, for instance, then a 201 g package of tomatoes can be used to make 19, 20, 21, or 22 times the recipe, but not any other multiple. So we can think of this package as the inclusive range [19, 22].

How can we find these ranges? Let Q be the amount in the package and R be the amount of that ingredient required by the base recipe. Then any allowable multiple m must satisfy both $0.9 \times R \times m < Q$ and $Q < 1.1 \times R \times m$. Simplifying and rearranging this, we get $(10/11) \times (Q/R) < m < (10/9) \times (Q/R)$; any integer value of m that satisfies this inequality is part of the range, and we can find the lower and upper bounds of this range in constant time. (Note that a range might not include *any* integer values, e.g., for R = 10 and Q = 15.) It is possible to use integer division here and avoid floating point numbers entirely.

In the Small dataset, there can be at most two ingredients. If there is one ingredient, the problem is very simple: each package with a non-empty range (as described above) forms a kit, and each package with an empty range does not.

What if there are two ingredients? Then we need to pair up the packages into kits, but we must be careful, since not any assignment will be optimal. For example, suppose our recipe requires 5 g of A and 10 g of B, and we have the following packages: 45 and 50 of A, and 110 and 111 of B. Converting these to ranges, we get [9, 10] and [10, 11] for A, and [10, 12] and [11, 12] for B. If we pair the [10, 11] package of A with the [10, 12] package of B, then we will be left with a [9, 10] package of A and an [11, 12] package of B, and those cannot be paired. However, we can make two kits by pairing the [9, 10] package of A with the [10, 12] package of B, and the [10, 11] package of A with the [11, 12] package of B.

We could solve the problem via bipartite matching, but it is not needed. Since there can be at most 8 packages of each ingredient in the Small, it is possible to use brute force to try all 8! possible matchings and find the largest number of kits that can be formed.

Large dataset

The following greedy strategy works for the Large, for any number of ingredients:

- Keep making the smallest possible multiple of the recipe that we can.
- Whenever we have a choice of packets, we choose one with the smallest upper end of the range. Since we are always making the smallest multiple we can, we do not care about the parts of the ranges up to and including that multiple. We only care about how large the upper ends get. Since the ranges are continuous, a range with a larger upper end is strictly more flexible for future pairings than a range with a smaller upper end.

This strategy would not necessarily be optimal for arbitrary ranges, but it is in this case thanks to the following property: if I_1 and I_2 are the ranges of valid multipliers for the same ingredient, and I_1 is included in I_2, then I_1 and I_2 have at least one endpoint in common. This is not hard to

prove: if I_i comes from an original packet of size S_i, if $S_1 \leq S_2$, then the lower bound of I_1 must be no greater than the lower bound of I_2. Otherwise, $S_1 > S_2$, so the upper bound of I_1 must be no less than the upper bound of I_2. Let us call this property 0.

We need some preliminary properties of our greedy algorithm before the main proof:

- *Property 1:* Whenever it chooses to discard a range, then there was no valid kit remaining to use that range for. This follows directly by the condition to discard ranges.
- *Property 2:* Whenever it chooses to use a set of ranges to form a kit, and M is the lower bound of the intersection of all those ranges, there is no other set of ranges left that could possibly make a kit with a multiplier strictly less than M. This follows directly from the order in which the ranges are considered.

Now we can combine all those properties and proceed by contradiction. Let M_1, M_2, ..., M_k be the smallest multipliers of the kits produced by our greedy algorithm, in non-decreasing order. Let N_1, N_2, ..., N_k, N_{k+1} be the smallest multipliers of an assignment that produces one more kit, in non-decreasing order.

- Case A: Let i be the minimum i for which $N_i < M_i$. By property 2, that can only happen if there is no set of ranges to make a kit with multiplier N_i. That means the greedy algorithm discarded such ranges or used them for some previous kit. Property 1 prevents the discarding. Property 0 and the processing order prevented the algorithm from using it before, because if a range R is chosen at some point, all other remaining ranges for that ingredient have an upper bound that is no less than R's, and therefore, such choice can't prevent a future kit for being formed; all other ranges are, at that point, at least as useful as R for larger multipliers.
- Case B: There is no such i. Again, the greedy algorithm must have some range missing not to be able to produce the kit with multiplier N_{k+1}. However, the same argument as in Case A using properties 1 and 0 shows that the wrong earlier decision that made that range not available couldn't have happened.

We can implement this strategy with a separate list for each ingredient, sorted first by nondecreasing lower end of range and then by nondecreasing upper end of range. We keep looking at the set of earliest ranges of the lists. If they all have an intersection, make them into a kit and remove them. Otherwise, remove the least useful range (the one with the smallest upper limit). We can speed this process up by using a priority queue, but with $\mathbf{N} \times \mathbf{P} \leq 1000$, such optimizations are not necessary.