Code Jam 2018 - Round 2

Analysis: Graceful Chainsaw Jugglers

We can identify each juggler by a pair of integers (b, r) where 0 ≤ b ≤ **B** and 0 ≤ r ≤ **R**. The pair (0, 0) does not identify a valid juggler, although we could assume that it does and, since we can always add (0, 0) to any valid configuration that does not contain it, just solve the modified problem and subtract 1 to obtain the final answer. We can then focus on finding the size of the largest subset of V = [0; **B**] × [0; **R**] such that the sum of the first component of each element is **B**, and the sum of the second component of each element is **R**. Let us call b,r-valid to a set of distinct pairs that fulfills the sum conditions for specific b and r.

Test set 1

For Test set 1, we can first notice that the size of V is small. Therefore, we can do a dynamic programming iterating through V and deciding, for each pair, whether to include it or not. As part of the state we have to also include how many blue and red jugglers we can still include. Therefore, we enumerate V in any order v_1, v_2, ..., $v_{(\mathbf{B}+1) \times \mathbf{R}+1}$, and then we compute a function f(i, b, r) defined as the size of the largest b,r-valid subset of $\{v_1, v_2, ..., v_i\}$.

We can see that f has also a recursive definition as follows:

- f(0, 0, 0) = 0.
- f(i + 1, b, r) = max(f(i, b, r), 1 + f(i, b - left(v_i), r - right(v_i))) for all b, r, i.

The definition above is incomplete because it's not checking for indefinitions. A simple solution is to just define every undefined case as -infinity, or for this particular problem, - **B** - **R**.

Memoizing this recursive definition leads to an implementation that takes time proportional to the size of the domain of f, which is $(\mathbf{B}+1)^2 \times (\mathbf{R}+1)^2$, which is likely fast enough to solve Test set 1. If your implementation and language of choice are both on the slow side, it's possible that you need some additional insight to make it through the time limit. One simple trick is that the function f actually solves all possible test cases, since the set V is fixed. So, you can reuse the memoization table for all test cases and save 99% of the work compared to solving each test case from scratch. You can even compute the entire table before submitting, and include it in your source code, although you should be wary of this strategy in general since it might cause your source file to exceed the 1 MB limit specified in our rules.

Test set 2

For Test set 2, we will definitely need the additional insight of not resetting the memoization table for each test case. However, the important additional insight is more specific to the problem.

First, let us define weak-b,r-valid sets as sets of distinct pairs of non-negative integers such that the sum of the values of the left sides of each pair is less than or equal to b, the sum of the values of the right sides of each pair is less than or equal to r. Of course, every b,r-valid set is also a weak-b,r-valid set, but the converse is not true.

One important property of weak-b,r-valid sets is that for fixed b and r, the size X of the largest weak-b,r-valid set is equal to the size Y of the largest b,r-valid set. We can prove that by noticing: (1) Y ≤ X because every b,r-valid set is a weak-b,r-valid set; and (2) X ≤ Y because if we let S be a weak-b,r-valid set of size X and then take an element (i,j) of S such that i is maximum, and j is maximum among those with maximum i, we can construct S' = S - {(i,j)} ∪ {(i

+ db, j + dr)} where db and dr are the difference between the sums of the left/right values of the pairs and b/r, respectively. By construction, S' is b,r-valid, and by the maximality of the choice of (i,j), (i + db, j + dr) is not in S - {(i,j)}, and thus, the sizes of S and S' are equal.

Let us now define minimal-weak-b,r-valid sets as weak-b,r-valid sets S such that for any (i,j) in S with i > 0, (i-1, j) is also in S, and analogously, for any (i,j) in S with j > 0, (i, j-1) is also in S. Similarly to the above, we can prove that the size of the largest minimal-weak-b,r-valid set for fixed b and r is the same as the size of the largest weak-b,r-valid sets by constructing a minimal-weak-b,r-valid sets of size X from a given weak-b,r-valid set S of size X. Let (i,j) in S be such that the minimality of the definition is violated (if such exists). Then, replace (i,j) in S for either (i-1,j) or (j-1,i) as appropriate to obtain another weak-b,r-valid set of the same size. This step strictly decreases the sum of the values of all pairs in S, thus, at some point, there are no such (i,j) to choose for and we successfully obtained a minimal-weak-b,r-valid set of size X.

The problem that remains is to find the size of the largest minimal-weak-b,r-valid set for given b and r. By the minimality condition, if (i,j) is in such largest set, so are (i-1, j), (i-2,j) (i,j-1), (i,j-2), (i-1, j-1), ... To formalize, (i,j) being present in a minimal-weak-b,r-valid set prescribes the presence of (i+1) × (j+1) pairs including itself. We can bound the sum of the left side values of all those pairs by (j+1) × (1 + 2 + ... + i) = (j+1) × i × (i+1) / 2, and analogously the sum of the right sides by (i+1) × j × (j+1) / 2. This means that instead of the set V from the solution above, we can use V', composed of the pairs in V that are under these bounds. This makes the size of V' be approximately $O(\mathbf{B}^{1/3} \times \mathbf{R}^{1/3})$ when **B** and **R** are close, which is a big reduction from the size of V which is $O(\mathbf{B} \times \mathbf{R})$. The overall complexity of the resulting algorithm for all test cases is then $O(\mathbf{B}^{4/3} \times \mathbf{R}^{4/3})$, which is fast enough to pass Test set 2.