Kick Start 2020 - Round F

Analysis: Yeetzhee

First, note that Pommel can always win the game in a finite number of moves on expectation. To do so for a given input sequence $\mathbf{A}_1$, $\mathbf{A}_2$, ... $\mathbf{A}_\mathbf{K}$, Pommel can simply reroll until the first $\mathbf{A}_1$ dice land on 1, the next $\mathbf{A}_2$ dice land on 2, and so on, with the last $\mathbf{A}_\mathbf{K}$ dice landing on **K**. Such an outcome satisfies the group arrangement required by the input; furthermore, on each roll, Pommel has a 1/**M** chance of rolling the needed value for the die she's on. This implies that Pommel is expected to win in **N**×**M** turns. Although this may not be optimal, it demonstrates that Pommel always has a winning strategy.

The basic procedure for calculating the expected value is the same for any approach. Suppose Pommel has already rolled and fixed some (potentially zero) dice. Furthermore, suppose she chooses a set S of dice configurations, such that she will not re-roll her current die if her dice configuration, after her current roll, is in S. If there is a p_i probability that her current dice configuration leads to the i-th element of S, then there is a $1 - \Sigma\, p_i$ probability that Pommel will have to re-roll. If we now let e_i equal the expected number of moves to win starting from state i, assuming Pommel plays optimally, we can solve for the expected number of moves to win starting from Pommel's current dice configuration. If we let this quantity be x:

$$x = 1 + (1 - \Sigma\, p_i)x + \Sigma\, p_i e_i$$

$$x = (1 + \Sigma\, p_i e_i) / (\Sigma\, p_i)$$

Approaches to the two tests sets differ in how they may enumerate the possible dice configurations and how they find the optimal set S for any given configuration.

Test set 1

For this test set it's sufficient to look at all dice roll results directly. There are at most $\Sigma_{0 \le i \le \mathbf{N}}\, \mathbf{M}^i < 60{,}000$ such possibilities. For a given configuration, performing another roll leads to one of **M** new dice configurations, so for each configuration we can loop through all $2^\mathbf{M}$-1 possible non-empty subsets of configurations that Pommel could choose to not re-roll on (ignoring subsets that contain configurations that make it impossible for Pommel to reach a winning configuration). For each choice of subset, we compute Pommel's expected turns until winning, and we take the subset with the lowest expected value.

We need to iterate over the dice configurations such that when we're computing the minimum expected value for a configuration C, we know the minimum expected values for each configuration that C can lead to. This can be done, for example, with DFS. The expected value from the empty configuration is our answer. This algorithm runs in $O(\mathbf{M}^\mathbf{N} \times 2^\mathbf{M} \times \mathbf{M})$ time which is sufficient.

As an example, consider the test case **N** = 3, **M** = 2, **K** = 2, $\mathbf{A}_1$ = 1, $\mathbf{A}_2$ = 2. Let's compute Pommel's expected turns from winning when she's rolled and locked down a 1. There are two possibilities for the dice configuration after her next turn: [1, 1] and [1, 2], each occuring with probability 1/2. As a precondition to computing the expected turns to win from [1], we know that, on expectation, it takes two moves to win from [1, 1] and one move to win from [1, 2]. We now consider the three possibilies for S: {[1, 1]}, {[1, 2]}, and {[1, 1], [1, 2]}.

S = {[1, 1]} leads to the equation $x = 1 + (1 - 0.5)\, x + 0.5 \cdot 2$, which results in x = 4.

S = {[1, 2]} leads to the equation $x = 1 + (1 - 0.5) x + 0.5 \cdot 1$, which results in $x = 3$.

S = {[1, 1], [1, 2]} leads to the equation $x = 1 + (1 - 0.5 - 0.5) x + 0.5 \cdot 1 + 0.5 \cdot 2$, which results in $x = 2.5$. This is the lowest value, so we now know that if Pommel has rolled a single 1, she will win in 2.5 more turns on expectation if she plays optimally.

Test set 2

For this test set, the number of possible dice roll outcomes, 50^{50}, is far too large to be enumerated in time. We instead focus on the groupings of dice that could possibly lead to the desired configuration. For example, if Pommel rolls [1, 1, 2, 2] or if she rolls [3, 4, 3, 4], either way, she'll have two groups of two dice, which are equivalent for this problem. A dice grouping can be expressed as a **K**-tuple of integers $(x_1, x_2, ..., x_{\mathbf{K}})$ such that the tuple elements are non-decreasing: $x_i \leq x_{i+1}$ for $1 \leq i < \mathbf{K}$. The i-th element of the tuple expresses the number of dice in the i-th smallest dice group, which is zero if there are fewer than **K** non-empty groups in the current dice configuration. For example, if **K**=3 and the dice results so far are [2, 2, 3, 2, 2, 3], the grouping can be expressed as (0, 2, 4).

We need to count how many possible **K**-tuples we need to consider to get an estimate of our algorithm's runtime. One simple bound is that we only need to consider a tuple T if the sum of T's elements is at most **K**. The total number of tuples we need to consider is therefore at most $\Sigma_{0 \leq i \leq \mathbf{K}}$ p(i) where p(i) is the number of **K**-tuples whose elements are non-decreasing and add to i. p(i) can be calculated with a script, but it is also the i-th partition number. The sum of the first 50 partition numbers is only slightly more than a million, which is very tractable.

A dice roll could lead Pommel from a given configuration to at most **K** new configurations. We now need a way of computing the optimal subset of new configurations S to not re-roll that is faster than the naive $O(\mathbf{K} \times 2^{\mathbf{K}})$ approach. Consider the expected value equation shown above. First, note that if for some i, $e_i > x$, removing configuration i from S will decrease the value of x. Similarly, if there is some configuration C with expected value $E[C] < x$ that is not included in S, adding it to S will lower x. This implies that if some configuration C with expected value E[C] is in S, then all states C' with $E[C'] < E[C]$ must also be in S if Pommel is playing optimally. We can therefore sort all configurations reachable from a given configuration so that they're in ascending order of expected turns until winning. It is guaranteed that the optimal S is a prefix of the sorted list, so it can be found by sorting and iterating over the list in $O(\mathbf{K} \log \mathbf{K})$ time.

Combining our improved optimal-subset routine with our bound for the number of **K**-tuples (which was in fact quite loose and can be shown to be significantly lower) gives us an algorithm that runs in time for $\mathbf{M}, \mathbf{N} \leq 50$.