Code Jam 2021 - Round 3

Analysis: Binary Search Game

Test Set 1

Let $f(k)$ be the number of ways in which the game ends with a score $\geq k$. Then, the number of ways in which the game ends with a score of exactly k is $f(k) - f(k+1)$. Therefore, the answer is

$$\begin{aligned}\sum_{k=1}^{\mathbf{M}} k(f(k) - f(k+1)) &= \sum_{k=1}^{\mathbf{M}} kf(k) - \sum_{k=1}^{\mathbf{M}} kf(k+1) \\ &= \sum_{k=1}^{\mathbf{M}} kf(k) - \sum_{k=2}^{\mathbf{M}+1} (k-1)f(k) \\ &= f(1) - \mathbf{M}f(\mathbf{M}+1) + \sum_{k=2}^{\mathbf{M}} (k-(k-1))f(k) \\ &= \sum_{k=1}^{\mathbf{M}} f(k).\end{aligned}$$

Notice in the last step that $f(\mathbf{M}+1) = 0$ by definition.

Let us now focus on calculating $f(k)$. For each card, we only care about whether its value is $\geq k$ or not. Let S be a fixed subset of cards which have value $\geq k$ (we will go through all $2^{\mathbf{N}}$ such subsets). We can assign a score of 0 (for cards outside of S) or 1 (for cards in S) to each consecutive subsequence of cells on the board that can be presented to players, in increasing order of length. Notice that only subsequences of cells whose length is a power of 2 and who are "aligned" can be presented to a player. That is, sequences containing cells of the form $i2^j + 1, i2^j + 2, \ldots, i2^j + 2^j$ for each j between 0 and $\mathbf{L}$, inclusive, and each i between 0 and $2^{\mathbf{L}-j} - 1$, inclusive. There are $2^{\mathbf{L}} + 2^{\mathbf{L}-1} + \cdots + 2 + 1 = 2^{\mathbf{L}+1} - 1$ valid combinations to go through. If we know the score for sequences of shorter lengths, we can decide the score for a sequence by checking whose turn it is (it is Alice's turn if and only if j and $\mathbf{L}$ have the same parity) and picking either the maximum or the minimum score of its two halves. If the score assigned by this to the full sequence is 0, then there is no way to finish with a score $\geq k$ with combination S. Otherwise, we need to compute the number of ways in which this can happen. This is simply the number of original card values that yield S, which is $(\mathbf{M} - k + 1)^t (k-1)^{\mathbf{N}-t}$ where t is the number of cards that are $\geq k$ in S.

Overall, this solution takes $O(\mathbf{M} \times 2^{\mathbf{N}} \times 2^{\mathbf{L}})$ time, which is fast enough to pass Test Set 1.

Test Set 2

Note that the expression $(\mathbf{M} - k + 1)^t (k-1)^{\mathbf{N}-t}$ from the previous solution, if we consider k the only variable, expands into a polynomial of degree at most $\mathbf{N}$. Consequently $f(k)$ is also a polynomial in k of degree at most $\mathbf{N}$ (as it is the sum of such polynomials).

If $f(k)$ is a polynomial of degree at most $\mathbf{N}$, then $g(x) = \sum_{k=1}^{x} f(k)$ is a polynomial in x of degree at most $\mathbf{N}+1$. This is a consequence of Faulhaber's formula.

Thus, the polynomial $g(x)$ can be fully determined by evaluating it at $\mathbf{N}+2$ different points, say $g(0), g(1), \ldots, g(\mathbf{N}+1)$. We can evaluate those values in the same way as Test Set 1 and then use interpolation to find $g(\mathbf{M})$.

This yields an algorithm that takes time $O(\mathbf{N} \times 2^{\mathbf{N}} \times 2^{\mathbf{L}})$ to evaluate $\mathbf{N}+2$ points of $f(k)$, plus the time complexity of evaluating $g(\mathbf{M})$ using interpolation. The latter can be done by evaluating the Lagrange polynomial in a straightforward way in $O(\mathbf{N}^2)$, and there are faster methods too, but this does not impact the overall runtime.

To speed up the calculation of $g(0), g(1), \ldots, g(\mathbf{N}+1)$, we find a way to reduce the $2^{\mathbf{N}}$ factor. Note that if a card's index appears only once (or not at all) in the board, we can do without fixing its value in advance (as being $\geq k$ or not). We can fix the values that are repeated in the board and then run the greedy algorithm but instead of scores 0 and 1 we compute, for each subsequence, the number of ways that it can be made 1 by assigning the values of cards whose indices are not repeated in the board. The decisions can still be made greedily, but instead of doing all the combinatorics at the end, we do them at each decision point.

With this optimization, we only need to run the greedy algorithm 2^X times per each value of g that we need, where X is the number of cards whose indices appear more than once in the board. There can be at most $2^{\mathbf{L}-1}$ of those cards, so we have reduced the time complexity of the first step to $O(\mathbf{N} \times 2^{2^{\mathbf{L}-1}} \times 2^{\mathbf{L}})$. This is finally enough to pass Test Set 2.