D - Dull Game

We notice that, at each step, if we choose the a point with x-coordinate x_0, then the maximum point can be obtained is y_0 where:

- If there exists segments that cover x_0, y_0 is the minimum height of all such segments.
- Otherwise, $y_0 = m$.

Thus, at each step, we only need to find the value of x_0 that results in the largest y_0.

This problem is a basic application of segment tree with lazy propagation. We need to construct a segment tree where each node stores the following value of some segment $[l, r] (0 \leq l \leq r \leq n)$: The highest point can possibly attained if we choose $x_0 \in [l, r]$. The update and get methods can be implemented with lazy propagation. You can look at the solution code for more detail.

At each step, updating takes $O(\log n)$ while querying takes $O(1)$. Thus, the total complexity is $O(n \log n)$.

I - Impossible Task

Consider $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ where $p_1, p_2, \cdots, p_k$ are prime factors of n. Then, the sum S of all factors of n can be calculated as

$$S = (1 + p_1 + \cdots + p_1^{\alpha_1})\,(1 + p_2 + \cdots + p_2^{\alpha_2}) \cdots (1 + p_k + \cdots + p_k^{\alpha_k}) \qquad (1)$$

Thus, we only need to compute p_i, α_i for all $1 \leq i \leq k$. Since $\sum_{i=1}^{k} \alpha_k \leq \lceil \log_2 n \rceil$, with all p_i, α_i provided, S can be calculated in $O(\log n)$.

To calculate all prime factors of n and their power quickly, we can use Erastothene Sieve with a modification. While running the sieve, for each composite number, we can store any of its prime number. Then, we can factorize any number n in $O(\log n)$ (see solution code for more details).

For each n, we need $O(\log n)$ time to answer the query, thus the total complexity is $O(q \log n)$. Note: Here we excluded the complexity of running the sieve since its complexity $O(N \log N)$ (where N is the maximum value of n), in this case, is a constant.

J - Just a hike!

We can solve the problem using binary search on the columns. Suppose we are searching in $[l, r]$. Let $mid = \lfloor \frac{l+r}{2} \rfloor$. Let

$$\begin{aligned}
h_{mid} &= argmax_{1 \le i \le m} h[i, mid] \\
h_l &= \begin{cases} argmax_{1 \le i \le m} h[i, mid-1] & \text{if } mid - 1 \ge l \\ -1 & \text{otherwise} \end{cases} \\
h_r &= \begin{cases} argmax_{1 \le i \le m} h[i, mid+1] & \text{if } mid + 1 \le r \\ -1 & \text{otherwise} \end{cases}
\end{aligned} \tag{2}$$

- If $h[1, h_{mid}]$ is a peak then return (mid, h_{mid}) as the result.
- If $h[1, h_{mid}] < h[1, h_r]$ (given that $h_r \neq -1$), there exists a peak in the segment $[mid + 1, r]$. Thus, we apply binary search on $[mid + 1, r]$.
- If $h[1, h_{mid}] < h[1, h_l]$ (given that $h_l \neq -1$), there exists a peak in the segment $[l, mid - 1]$. Thus, we apply binary segment on $[l, mid - 1]$.

We starts the search with segment $[1, n]$.

At each step, we need to calculate all the values in 3 columns $mid-1, mid, mid+1$. This can be done in $O(m)$. Thus, the total complexity is $O(m \log n)$.

F - Featured Animals

Placing animals from left to right is a Markov process, so we can use matrix multiplication. Let state vector have ith element representing the number of configurations when the current cage contains type i animal. Transition matrix is the one matrix except where corresponding adjacent types are forbidden, in which case the matrix element is zero. We must zero out some state vector elements in certain stages because these cages forbid some types of animals. Time complexity is $O(NM^2)$.

We can optimize with fast matrix exponentiation. This will give us time complexity $O(M^3 K \log \frac{N}{K}) = O(M^3 K \log N)$. We can further precompute base matrices, yielding time complexity $O(M^3 \log N + M^2 K \log \frac{N}{K}) = O((M + K)M^2 \log N)$.