Stylemaxxing

For a stylish haircut, note that if $\text{dist}((r,c),(i_1,j_1)) = \text{dist}((r,c),(i_2,j_2))$, then $a_{i_1,j_1} = a_{i_2,j_2}$, since we can apply the inequality in both directions. Let's denote by S_d all cells at distance d from (r,c) and let D be the maximum distance from (r,c) to any cell on the grid. Thus, a haircut is stylish if and only if the following two conditions hold:

- For every d, there exists $1 \le t_d \le n \cdot m$ such that $\forall (i,j) \in S_d : a_{i,j} = t_d$.
- For every two consecutive layers S_d and S_{d+1} we must have $t_d \ge t_{d+1}$.

Note that we can always make n horizontal cuts through each row with target length $x = 1$ and achieve a hairstyle where all hair have length 1, hence, making it a stylish haircut. Analogously, We can make m vertical cuts and obtain a stylish haircut. This means that the answer is at most $\min(n,m)$. Additionally, making cuts only with $1 \le x \le n \cdot m - 1$ is sufficient.

Group 1 $n = 1$ — 4 points

Since $\min(n,m) = 1$, the answer must be either 0, or 1. Thus, we just need to check if the initial haircut is already stylish, which can be done in $O(m^2)$ time by directly checking all pairs of hairs strands.

Group 2 $n \le 4, m \le 4, a_{i,j} \le 2$ — 8 points

Note that the only viable cut length is $x = 1$, so we have at most $\frac{n(n+1)}{2} \cdot m + \frac{m(m+1)}{2} \cdot n - n \cdot m \le 64$ different cuts to make. Since the answer is at most 4, we can iterate over all $C_{64}^1 + C_{64}^2 + C_{64}^3 = 43744$ possible cuts and check if they make the haircut stylish.

Group 3 $n \le 3, m \le 25$ — 11 points

Since the answer is at most 3, we just need to check if it is equal 0, 1, or 2. A slightly optimized $O(m^3)$ solution that iterates over all pairs of cuts and values of cuts should pass this subtask.

Group 4 $r = 1, c = 1, n \le 50, m \le 50, a_{i,j} \le 2$ — 15 points

Let's iterate over $\overline{d}$ — distance, at which we make the difference between cells equal to 2 and cells equal to 1, i.e., $t_0 = t_1 = \ldots = t_{\overline{d}} = 2$ and $t_{\overline{d}+1} = \ldots = t_D = 1$. Since $a_{i,j} \le 2$ in this subtask, we should not make any operations on any layers up to $\overline{d}$ and just need to check if they all consist only of values 2. In order to calculate the minimum number of operations required to trim layers from $\overline{d}+1$ onward to length 1, we come to the first main idea of the solution:

Layers $S_{\overline{d}+1}, \ldots, S_D$ represent a convex connected area, hence, each operation we make inside these layers can be extended till the borders of this area. So we need to find the minimum number of rows and columns so that every cell with $a_{i,j} = 2$ is covered by at least one row or column.

This problem can be translated into graph theory: Make a bipartite graph with left and right parts corresponding to rows and columns of the grid, while an edge between a left vertex u_i and a right vertex v_j exists as long as $a_{i,j} = 2$. The minimum number of rows and columns required to cover all cells with $a_{i,j} = 2$ is exactly the minimum vertex cover of this graph. According to Konig's theorem, the size of the minimum vertex cover in a bipartite graph is equal to the number of edges in maximum matching of this graph, which can be found using Kuhn's algorithm.

To sum up, we iterate over $\overline{d}$ — the border of values 2 and 1, build a bipartite graph, and find the maximum matching of this graph to get the final answer. Since Kuhn's algorithm's runtime is $O((n+m)nm)$, total runtime would be $O((n+m)^2 nm)$.

Group 5 $r = 1, c = 1, n \leq 25, m \leq 25$ — 10 points

To solve this subtask, we need to come up with the second main idea: $dp_{d,val}$ — minimum number of operations required to make layers $S_0, \ldots, S_d$ stylish using operation only on them so that $t_d = val$. To compute this dp, we need to iterate over the layer $k \leq d$, which is chosen to satisfy $t_{k-1} > t_k = t_{k+1} = \ldots = t_d$ (and $t_d > t_{d+1}$ in accordance to $t_{k-1} > t_k$). With this conditions on k, we can surely state that all of the operations done on layers $S_k, \ldots, S_d$ do not contain any cell from layers $S_0, \ldots, S_{k-1}$. We can also state that they don't contain any cells from layers $S_{d+1}, \ldots, S_D$, since they are useless for them. Since layers $S_k, \ldots, S_d$ still form a convex area, we still can construct a bipartite graph and find maximum matching on it to complete the transition $dp_{k-1,\widehat{val}} \to dp_{d,val}$. We actually don't need the exact value $\widehat{val}$, except for it being strictly greater than val, thus, it can be optimized out using suffix minimums.

Runtime of this algorithm is $O((n+m)nm)$ number of dp states, $O(n+m)$ transitions, and each transition is done using Kuhn's algorithm in $O((n+m)nm)$ time, so it would be $O((n+m)^3n^2m^2)$ in total. Since Kuhn actually works way faster than $O((n+m)nm)$, such solution passes the subtask without any problems.

Group 6 $n \leq 25, m \leq 25$ — 20 points

This subask differes from the previous only in the fact that the area covered by layers $S_k, \ldots S_d$ is not convex. To deal with this, we simply need to cut every row and column that intersects inner layers $S_0, \ldots S_{k-1}$ in two and do the same as in subtask 5.

Group 7 $n \leq 75, m \leq 75$ — 16 points

To solve this subtask, we need to notice that the only useful value of val is the minimum value of $a_{i,j}$ among $S_0, \ldots, S_d$ (it definitely cannot be larger than this and it is useless to make it smaller than this). This reduces total runtime by a factor of nm.

Group 8 no additional constraints — 16 points

To solve this subtask, we need to start computing the dp from current layers onward, i.e., fix k and iterate over $d = k, k+1, \ldots, D$ to calculate the transition $dp_{k-1} \to dp_d$. Let's look at how the bipartite graph required for calculating transitions evolves when we increase d. The splitted rows and columns remain the same, while new rows and columns might be added. The target value val can only decrease, so the edges from previous layers must stay in the graph; while some new cells from current layer S_d or previous layers might be added as new edges to the graph. This means that the graph from $d-1$ is a subgraph of the graph from d and we can reuse the computed maximum matching saved for $d-1$ and just try augmenting it using the newly added edges.

This sums up in the following algorithm: iterate dp from $k = 0$ to $k = D$ in this order. Iterate over $d = k$ to $d = D$ while maintaining the current graph and its maximum matching to accurately calculate the transition $dp_{k-1} \to dp_d$. Total runtime of this algorithm would be $O((n+m)^2nm)$. However, there is a proven runtime for Hopcroft-Karp's algorithm (optimized for bipartite graphs version of Dinic's maximum flow algorithm) to be $O(|E|\sqrt{|V|})$ for a dynamically increasing graph, so the runtime becomes $O((n+m)nm\sqrt{n+m})$. In practice, Kuhn's and Hopcroft-Karp's algorithms work almost the same on dynamically increasing almost complete graphs, so it was sufficient to use Kuhn's algorithm to pass all tests.