Rect (Review)

Subtasks 1, 2, and 3

Subtask 1 can be solved via a trivial $\mathcal{O}(n^6)$ solution: considering all $\mathcal{O}(n^4)$ rectangles and checking whether each of them is valid in $\mathcal{O}(n^2)$. Moreover, if we use breaks in loops for checking whether a rectangle is valid, then this solution is accepted in Subtask 1, 2, and 3. Moreover, an $\mathcal{O}(n^5)$ solution exist which gets accepted in Subtask 1 and 2, and an $\mathcal{O}(n^4)$ solution exist which gets accepted in Subtask 1, 2, and 3.

Subtask 5

In this subtask, valid rectangles are all consecutive subsets of the middle. All of these rectangles can be checked in time $\mathcal{O}(m^2)$.

Subtask 6

This subtask is finding a 0 rectangle with 1 on outer borders (except corners). This is a classic dynamic programming problem which can be solved in $\mathcal{O}(n^2)$. Moreover, most of the $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2 \log n)$ solutions run in $\mathcal{O}(n^2)$ in this subtask.

Subtasks 4 and 7

Solution 1

We denote a pair of cells in a row or a column which are on the outer border of a valid rectangle as a good pair. The total number of good pairs are bounded by $\mathcal{O}(n^2)$ and can be found using a stack and traversing over cells of each row and each column. Then, we can extend these good pairs into two potential sides of a valid rectangle. Therefore, we have potential left and right sides, and potential top and bottom sides of valid rectangles. Finally, we try all cells as the bottom-right corner of valid rectangles and match sizes. If the matching phase is done naively, we have a $\mathcal{O}(n^3)$ solution which get accepted in subtask 4 but not Subtask 7. However, many optimizations results in getting accepted in Subtask 7. The best of which is using a Fenwick tree which decreases the total running time downto $\mathcal{O}(n^2 \log n)$. Moreover, in order to get accepted in Subtask 7, one might not use time-consuming data

structure libraries such as STL maps.

Solution 2

A very useful observation is that in a good pair, if we look at the smaller element, the pair is unique (one for the left direction, and one for the right direction). Using this observation, we can change the data structure needed for keeping good pairs into two simple arrays. For a valid rectangle, let a block corner be a 2x2 square that exactly one of its elements is inside the rectangle. Each valid rectangle have exactly four block corners. We can connect block corners of a valid rectangle which are in a row or a column based on whether which one captures the larger element in the outer border of the row or column. These connections define a structure for a valid rectangle. At most 2^4 = 16 structures exist and for each structure there is a block corner that uniquely define the valid rectangle. Therefore, the number of valid rectangles is bounded by $\mathcal{O}(n^2)$. After finding these $\mathcal{O}(n^2)$ candidate rectangles, we can check each of them in $\mathcal{O}(1)$ after a preprocess in $\mathcal{O}(n^2)$.