Kick Start 2019 - Round F

Analysis: Flattening

Test set 1 (Visible)

We can start with an interesting observation, whenever we rebuild a section of the wall, it is equivalent to removing that section. So we can follow two steps,

1. Pick a set of sections of the wall to remove.
2. Check if in the remaining wall sections, the number of positions where $\mathbf{A}_i \neq \mathbf{A}_{i+1}$ are less than or equals **K**.

There are $2^{\mathbf{N}}$ possible sets of sections, and for each set, checking if the remaining sections will make Blotch happy or not will take O(**N**) time. So the time complexity of this approach is $O(2^{\mathbf{N}} \times \mathbf{N})$, which is fast enough for test set 1.

Test set 2 (Hidden)

We need to approach the problem a bit differently for test set 2. We can start with observing the amount of walls we need to remove for a particular range of consecutive wall sections[i...j]. The most optimal way of making a set of consecutive wall sections have same height would be to figure out which height appeared the most, and remove all sections with a different height.

So for each consecutive set of walls, we can calculate number of wall removals needed. Let's say, R(i, j) defines number of removals necessary to have all wall sections from i to j have the same height.

We can define a solution based on a function F(x, k), where F(x, k) denotes the number of the walls we need to remove so that the sections from 1 to x in the input has $\mathbf{A}_i \neq \mathbf{A}_{i+1}$ in at most k places. We can define the recurrence relation as, F(x, k) = min(F(i, k-1) + R(i+1, x)) for all $1 \le i \le x-1$. The minimum number of walls that we need to remove for given **N** and **K** is F(**N**, **K**). We can compute this function using dynamic programming, which will have $O(\mathbf{N}^2 \times \mathbf{K})$ time complexity, which is fast enough for test 2.

We can have a faster solution, if we decompose the problem from a slightly different angle. We can think about running binary search on number of removals needed to satisfy the condition. In each iteration of binary search, we need to calculate if it is possible to have k or less number of positions where $\mathbf{A}_i \neq \mathbf{A}_{i+1}$ if we have at most x removal, x being the value of binary search value of that iteration. That can be also calculated with another dynamic programming, leading us to a solution with time complexity of $O(\mathbf{N}^2 \times \log(\mathbf{N}))$.