Kick Start 2020 - Round C

Analysis: Stable Wall

Test set 1

Each of the polyominos contains only one letter. We can take each of the permutations of the distinct letters, and check if the resulting wall is stable by following the order of the letters in the permutation to place the polyominos, one by one. There are **N** distinct letters in the input, so we will have **N**! permutations.

Let's say W is the given input wall letters. There are various ways to check if a wall is stable for a given permutation. One way is to create a new 2D array of integers, I. I[i][j] will contain the index of the letter W[i][j] in the permutation. Each of the letters at(i, j) in the wall must be placed after the letter below them, i.e. at(i+1, j), if they are not part of the same polyomino. We can check if it's true by checking if I[i][j] ≥ I[i+1][j], for all positions (i, j). Since I contains relative indexing of the order of the polyominos that were placed, this check works as intended. We can check this in O(**R** × **C**) time, so the runtime complexity of this approach for each test case is O(**N**! × **R** × **C**). This is sufficient for test set 1.

Test set 2

Given the picture of the wall, if two polyominos are positioned one above another, and they touch each other, then to make a stable wall, the bottom one must be placed before the top one. This gives us a set of orderings of pairs of letters which must be followed while placing the polyominos.

We can make a graph with the set of orderings just mentioned. Each polyomino/letter can be considered as a vertex, and each of the orderings can be considered as a directed edge of the graph. If the resulting graph contains any cycle, it is not possible to have a stable wall. Otherwise, we can use topological sort to find a suitable order that fulfills all the orderings. We can find all the orderings in O(**R** × **C**) time. We can find a topological sorting of the vertices of a directed acyclic graph by using depth first search. Depth first search gives us a runtime complexity of O(number of edges + number of vertices). The number of edges in the graph is at most (**R-1**) × **C**, as each cell of the grid adds at most one edge, to the cell right above it. So the total runtime complexity of this approach for each test case is O(**R** × **C**) + O(**N** + **R** × **C**) = O(**R** × **C**), since **N** can be at most **R** × **C**. This is sufficient for test set 2.