CodeForces Round 1037 Div. 2D / Div. 1B

omeganot

July 20, 2025

For a valid path to exist, the Manhattan distance between all p_i and p_{i+1} must be 2. Between any p_i and p_{i+1} there are at most two possible intermediate squares. If p_i and p_{i+1} are diagonally adjacent, there are two possible intermediate squares, and otherwise (there is one square between them) there's exactly one.

We can think of the problem as a matching problem. We can create a bipartite graph where the nodes are the cells of the conveyor. The squares $p_1 \dots p_k$ are colored white and all the possible intermediate squares are colored black. We can then draw an edge between nodes p_i and an intermediate square if that square connects p_i and p_{i+1}. We want to count the number of matchings on this graph that saturate all k of our white nodes.

We can calculate the answer for each connected component separately. Our answer will be the product of these results.

Let's now look at an arbitrary connected component. If the number of white nodes exceeds the number of black nodes, we know that the answer is 0, since we can't match all white nodes. I was able to find 3 cases with a lot of inspection and example cases.

Firstly, if the connected component is a tree and the number of white nodes equals the number of black nodes, the answer is 1. Every white node and every black node will be paired. Thus, if we have such a connected component, each leaf must be paired with its parent. We can then remove these leaves and the result will be yet another connected component which is a tree with equal number of white and black nodes. If we repeat the process, we can show that there is only one matching.

It is also possible for a connected component to be a tree but with one more black node than white nodes. Note that any more is not possible, as any valid connected component can be constructed with the following algorithm:

First, begin with a white node and its neighboring black nodes (there will

either be one or two). Then, add a new white node, connect it to an existing black node, and then either connect it to a new black node, an existing black node, or none at all. We can see that each step always adds one more white node but at most one more black node. Thus, there can be at most one more black node than white nodes. In this scenario, let b be the number of black nodes. Our answer for this component ends up being b.

Any valid matching will leave out one black node. We can show that for each black node, there is a corresponding matching with that node removed. It's important to note that for there to be more black nodes than white nodes, each white node is connected to two black nodes (we can use our constructive argument yet again; to we must always add one white node and one black node each step to maintain more black nodes than white nodes). Thus, let's say we remove an arbitrary black node. For each white node that was previously connected to it, they must have another neighboring black node, and we must match with that black node. Then, we can remove the matched black nodes. Once again, any neighboring white nodes must have another match, and we can continue the process until no nodes remain. Thus, there is a bijection between the black nodes of a valid component and the number of matchings that exhaust the white nodes.

Lastly, there is the case where our connected component is not a tree. If there is a cycle, then the number of black nodes cannot exceed the number of white nodes, since at some point in our construction we must have created one new white node but no new black nodes. Thus, the number of white nodes equals the number of black nodes, and our graph is a cycle with potentially some trees hanging off. Since each node will be paired, we can still use the idea to match each leaf with its parent. However, we will end up with a cycle that is still unmatched. Trivially, the cycle has exactly two possible matchings, so the answer is 2.

To create the connected components, I use a DSU. We will need to store the number of white nodes and black nodes in each component. Furthermore, to check if a component is a cycle, we will abuse the idea that a connected graph where the number of nodes doesn't exceed the number of edges will have at least one cycle, so we must maintain the number of edges in each component as well. Our final Time Complexity is $O((nm + k)\alpha(nm))$