Code Jam 2021 - Round 2

Analysis: Retiling

The first simple observation we need to solve this problem is that we never swap two tiles that are showing the same color.

The main not-that-simple observation is that any given tile should either be flipped or swapped (or neither), but never both, regardless of costs. This can be proved by considering all operations that affect tile a: if there is a swap of a and b followed by a flip of a or vice versa, the same effect can be achieved for less cost by simply flipping b (which may have moved by its own swaps in between).

A small addition to the above is that we never want to flip the same tile more than once. Since each tile is only affected by one type of operation, we can assume that all swaps happen before all flips. Thus, after all swaps are done, the flips are fixed: any tile that has a different color than the target state must be flipped, and the other tiles must not.

Test Set 1

In Test Set 1, we can make the additional observation that no tile is to be swapped twice. Since we should not swap tiles when they are showing the same color, swapping a and b and then b and c is equivalent to flipping a and c, which costs the same. Because we swap each tile only once, we must only swap two tiles if the swap fixes both of their colors (the samples already hint at this fact). Since swaps fix colors for two tiles and flips for one tile, we want to do as many swaps as we can within this restrictions, and then flip the rest.

At this point, the problem becomes a classic one: some cells of the matrix need to be changed, and we want to match as many of those as possible with an adjacent cell that also needs to be changed, but to the other color. Since the graph of cells and adjacencies is bipartite, this can be solved with a maximum matching algorithm on a bipartite graph. The solution is then the number of cells that need to change minus the size of such a maximum matching.

Test Set 2

As the additional sample shows, in Test Set 2 it is very possible that we need to swap the same tile multiple times. Luckily, we can generalize the solution explained for Test Set 1 for this scenario.

When a tile participates in multiple swaps, it effectively executes a swap with another tile that is not necessarily adjacent. While the other tiles are displaced by these operations, they are all of the same color, so that displacement has no effect. To put it in a different way, we can look at a swap as moving an `M` to an adjacent cell, and consecutive swaps are just consecutive moves.

Assume without loss of generality that the number of `M`s increases between the starting state and the target state (otherwise, use `G`s instead). Since we said we do all swaps first, that means that we move some of the `M`s that need to switch to `G`s to positions that have `G`s that need to be `M`s. We can do that by matching those two types of cell (regardless of adjacency). Any unmatched `G` into `M` position needs to be flipped (the number of these is fixed by the input).

For each matched pair, the cost of fixing both positions is either the minimum orthogonal distance between the two positions multiplied by $\mathbf{S}$ or $2\mathbf{F}$, since we can also do two flips.

The reasoning above basically built a weighted bipartite matching between cells that start at `M` and need to be turned into `G` and cells that start at `G` and need to be turned into `M`. The total cost is the cost of the maximum matching of minimum weight plus $\mathbf{F}$ times the number of unmatched cells. We can apply any minimum cost maximum matching algorithm for bipartite graphs like the Hungarian algorithm.