Problem Set 3 - Dynamic Programming

Problem C

Before even going into this question I had asked my friend about it, and they had pointed out it was a three-dimensional DP problem, which was, at that point, a little bit of a relief given my difficulties with implementing the exponential DP in question 2. I think the hint did give away quite a bit about the question in that it cut out a lot of the lost fumbling at the beginning, I just needed to figure out what would make up our three dimensions in the state.

I was pretty certain that we were iterating through and trying to find the minimum cost of colouring 1-n trees, and thus, how many trees we have (considered) colouring would be a parameter. The next was likely which tree we coloured last, as we would need that information (from previous and current states) to calculate the beauty of each state. But it did take me a little bit longer to realise that beauty was the last state. I don't quite remember what the alternative theories I had were, as in retrospect, it is quite clear the beauty would be the last state, it just felt at the time, that beauty was more of an outcome (result of the DP) rather than a parameter in the DP.

After coming to the final state d[i][j][b] however, it was not too difficult to link them to previous states, and then code up the question (much easier to code than exponential dp at this time…). For each tree (i in n), we would consider all the colours that it could be painted, or keep its existing colour if it's already coloured. The state dp[i][j][b] represents the minimum cost to color trees from 1 to i, with tree i having color j, and the resulting beauty being b.

The transitions to dp[i][j][b] depend on the previous tree, its color (prev), and the beauty (b') up to that point: If the current color j is the same as the previous tree's color (prev), the beauty would remain unchanged. The cost in this case is the value of the previous state dp[i-1][prev][b], plus the cost of painting the current tree if it's uncolored. If the tree is already colored j, the cost is zero.

If the current color j is different from the previous tree's color (prev), the beauty increases by 1. The cost in this case is the value of the previous state dp[i-1][prev][b-1], plus the cost of painting the current tree if it's uncolored. Again, if the tree is already colored j, there's no additional cost.

Finally, the base case would just handle all for the first tree, if it's uncoloured, initialise dp[1][j][1] for all colors j with the respective painting costs of the first tree. If it's already colored, we only initialise dp[1][c1][1] (where c1 is its initial color) with a cost of zero.

After processing all the trees, the result is the minimum value in dp[n][j][k] across all colors j, since we want exactly k beauty groups.

Done deal :)