Paint House II

There are a row of *n* houses, each house can be painted with one of the*k* colors. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a `n x k` cost matrix. For example, `costs[0][0]` is the cost of painting house 0 with color 0; `costs[1][2]` is the cost of painting house 1 with color 2, and so on... Find the minimum cost to paint all houses.

Note:
All costs are positive integers.

Follow up:
Could you solve it in *O*(*nk*) runtime?

Solution 1

The idea is similar to the problem Paint House I, for each house and each color, the minimum cost of painting the house with that color should be the minimum cost of painting previous houses, and make sure the previous house doesn't paint with the same color.

We can use min1 and min2 to track the indices of the 1st and 2nd smallest cost till previous house, if the current color's index is same as min1, then we have to go with min2, otherwise we can safely go with min1.

The code below modifies the value of costs[][] so we don't need extra space.

```
public int minCostII(int[][] costs) {
    if (costs == null || costs.length == 0) return 0;

    int n = costs.length, k = costs[0].length;
    // min1 is the index of the 1st-smallest cost till previous house
    // min2 is the index of the 2nd-smallest cost till previous house
    int min1 = -1, min2 = -1;

    for (int i = 0; i < n; i++) {
        int last1 = min1, last2 = min2;
        min1 = -1; min2 = -1;

        for (int j = 0; j < k; j++) {
            if (j != last1) {
                // current color j is different to last min1
                costs[i][j] += last1 < 0 ? 0 : costs[i - 1][last1];
            } else {
                costs[i][j] += last2 < 0 ? 0 : costs[i - 1][last2];
            }

            // find the indices of 1st and 2nd smallest cost of painting current
house i
            if (min1 < 0 || costs[i][j] < costs[i][min1]) {
                min2 = min1; min1 = j;
            } else if (min2 < 0 || costs[i][j] < costs[i][min2]) {
                min2 = j;
            }
        }
    }

    return costs[n - 1][min1];
}
```

written by jeantimex original link here

Solution 2

maintain the minimum two costs min1(smallest) and min2 (second to smallest) after painting i-th house.

```
int minCostII(vector<vector<int>>& costs) {
    int n = costs.size();
    if(n==0) return 0;
    int k = costs[0].size();
    if(k==1) return costs[0][0];

    vector<int> dp(k, 0);
    int min1, min2;

    for(int i=0; i<n; ++i){
        int min1_old = (i==0)?0:min1;
        int min2_old = (i==0)?0:min2;
        min1 = INT_MAX;
        min2 = INT_MAX;
        for(int j=0; j<k; ++j){
            if(dp[j]!=min1_old || min1_old==min2_old){
                dp[j] = min1_old + costs[i][j];
            }else{//min1_old occurred when painting house i-1 with color j, so it cannot be added to dp[j]
                dp[j] = min2_old + costs[i][j];
            }

            if(min1<=dp[j]){
                min2 = min(min2, dp[j]);
            }else{
                min2 = min1;
                min1 = dp[j];
            }
        }
    }

    return min1;
}
```

written by kelly original link here

Solution 3

Explanation: dp[i][j] represents the min paint cost from house 0 to house i when house i use color j; The formula will be dp[i][j] = Math.min(any k!= j| dp[i-1][k]) + costs[i][j].

Take a closer look at the formula, we don't need an array to represent dp[i][j], we only need to know the min cost to the previous house of any color and if the color j is used on previous house to get prev min cost, use the second min cost that are not using color j on the previous house. So I have three variable to record: prevMin, prevMinColor, prevSecondMin. and the above formula will be translated into: dp[currentHouse][currentColor] = (currentColor == prevMinColor? prevSecondMin: prevMin) + costs[currentHouse][currentColor].

```
public class Solution {
public int minCostII(int[][] costs) {
    if(costs == null || costs.length == 0 || costs[0].length == 0) return 0;

    int n = costs.length, k = costs[0].length;
    if(k == 1) return (n==1? costs[0][0] : -1);

    int prevMin = 0, prevMinInd = -1, prevSecMin = 0;//prevSecMin always >= prevM
in
    for(int i = 0; i<n; i++) {
        int min = Integer.MAX_VALUE, minInd = -1, secMin = Integer.MAX_VALUE;
        for(int j = 0; j<k;  j++) {
            int val = costs[i][j] + (j == prevMinInd? prevSecMin : prevMin);
            if(minInd< 0) {min = val; minInd = j;}//when min isn't initialized
            else if(val < min) {//when val < min,
                secMin = min;
                min = val;
                minInd = j;
            } else if(val < secMin) { //when min<=val< secMin
                secMin = val;
            }
        }
        prevMin = min;
        prevMinInd = minInd;
        prevSecMin = secMin;
    }
    return prevMin;
}
}
```

written by Tracy123 original link here