> PS：本题是 LeetCode VIP 题型

题目介绍[1]

假如有一排房子，共 n 个，每个房子可以被粉刷成 k 种颜色中的一种，你需要粉刷所有的房子并且使其相邻的两个房子颜色不能相同。

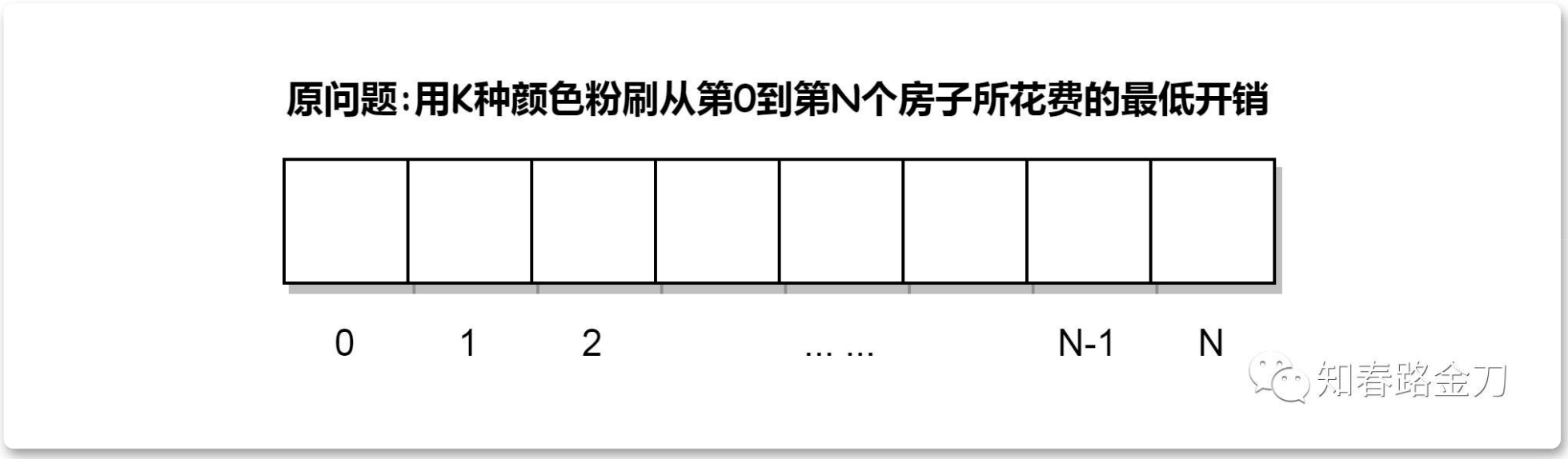
当然，因为市场上不同颜色油漆的价格不同，所以房子粉刷成不同颜色的花费成本也是不同的。每个房子粉刷成不同颜色的花费是以一个 **「n x k」** 的矩阵来表示的。

例如，**「costs[0][0]」** 表示第 0 号房子粉刷成 0 号颜色的成本花费；**「costs[1][2]」** 表示第 1 号房子粉刷成 2 号颜色的成本花费，以此类推。请你计算出粉刷完所有房子最少的花费成本。

注意：所有花费均为正整数。

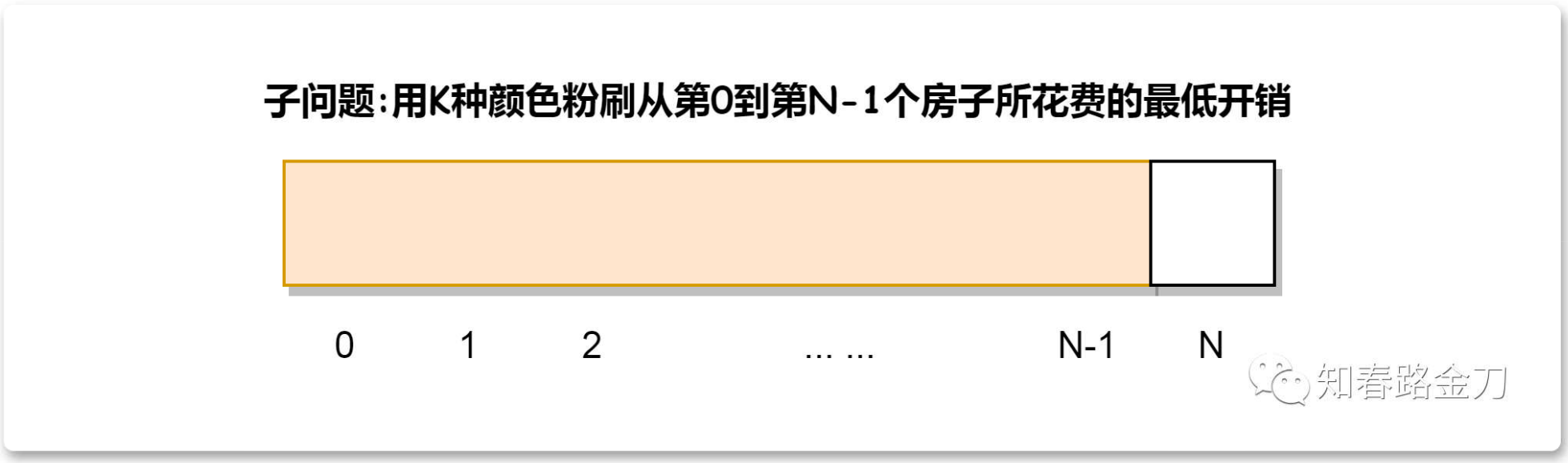
题目解答

这道题是**粉刷房子I的高阶版**，解答本题之前建议大家先看看粉刷房子 I。



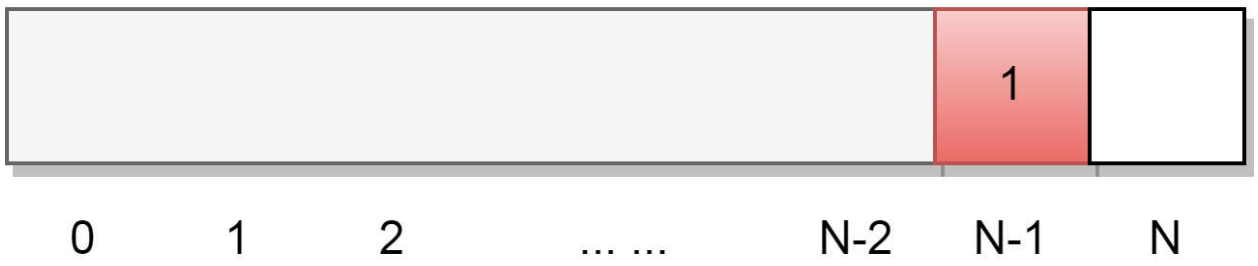
题目的原问题是求解用**「K 种颜色粉刷从第 0 到第 N 个房子所花费的最低开销」**，这个问题可以拆成如下 N 个子问题

- 用 K 种颜色粉刷第 0 个房子所花费的最低开销
- 用 K 种颜色粉刷从第 0 到第 1 个房子所花费的最低开销
- ... ...
- 用 K 种颜色粉刷从第 0 到第 N-2 个房子所花费的最低开销
- 用 K 种颜色粉刷从第 0 到第 N-1 个房子所花费的最低开销



注意这题有一个限制就是**「相邻的两个房子颜色不能相同」**，所以小粉刷匠每刷一个房子的时候，他需要思考这个房子要刷哪种颜色，刷第 1 种颜色？刷第 2 种颜色？刷第 K 种颜色？这样每一个子问题又可以继续拆解变成如下 K*N 个子问题

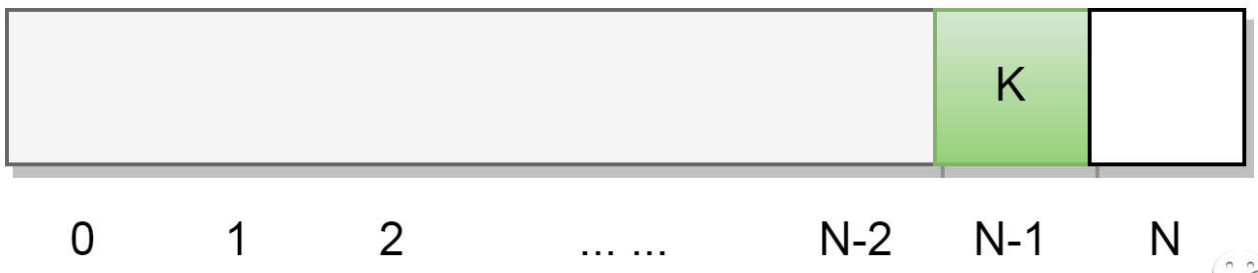
拆解子问题1:用K种颜色粉刷从第0到第N-1个房子所花费的最低开销
(其中第N-2个房子不能是第1中颜色，因为第N-1个房子要刷第1种颜色)



拆解子问题2:用K种颜色粉刷从第0到第N-1个房子所花费的最低开销
(其中第N-2个房子不能是第2种颜色，因为第N-1个房子要刷第2种颜色)

... ...

拆解子问题K:用K种颜色粉刷从第0到第N-1个房子所花费的最低开销
(其中第N-2个房子不能是第K种颜色，因为第N-1个房子要刷第K种颜色)



- 用 K 种颜色粉刷第 0 个房子所花费的最低开销
  - 给第 0 个房子刷第 1 种颜色时，粉刷从第 0 到第 0 个房子的最低开销是多少？
  - 给第 0 个房子刷第 2 种颜色时，粉刷从第 0 到第 0 个房子的最低开销是多少？
  - ... ...
  - 给第 0 个房子刷第 K 种颜色时，粉刷从第 0 到第 0 个房子的最低开销是多少？
- 用 K 种颜色粉刷从第 0 到第 1 个房子所花费的最低开销
  - 给第 1 个房子刷第 1 种颜色时，粉刷从第 0 到第 1 个房子的最低开销是多少？
  - 给第 1 个房子刷第 2 种颜色时，粉刷从第 0 到第 1 个房子的最低开销是多少？
  - ... ...
  - 给第 1 个房子刷第 K 种颜色时，粉刷从第 0 到第 1 个房子的最低开销是多少？
- ... ...
- 用 K 种颜色粉刷从第 0 到第 N-2 个房子所花费的最低开销
  - 给第 N-2 个房子刷第 1 种颜色时，粉刷从第 0 到第 N-2 个房子的最低开销是多少？

    - 给第 N-2 个房子刷第 2 种颜色时，粉刷从第 0 到第 N-2 个房子的最低开销是多少？
    - … …
    - 给第 N-2 个房子刷第 K 种颜色时，粉刷从第 0 到第 N-2 个房子的最低开销是多少？
- 用 K 种颜色粉刷从第 0 到第 N-1 个房子所花费的最低开销
    - 给第 N-1 个房子刷第 1 种颜色时，粉刷从第 0 到第 N-1 个房子的最低开销是多少？
    - 给第 N-1 个房子刷第 2 种颜色时，粉刷从第 0 到第 N-1 个房子的最低开销是多少？
    - … …
    - 给第 N-1 个房子刷第 K 种颜色时，粉刷从第 0 到第 N-1 个房子的最低开销是多少？

确定状态转移方程

子问题已经确定出来了，那么如果我们知道了**「用 K 种颜色粉刷从第 0 到第 N-1 个房子 K 种颜色所花费的最低开销」**，那么我们如何根据这个子问题来算出原问题**「用 K 种颜色粉刷从第 0 到第 N 个房子所花费的最低开销」**呢？

粉刷匠为了让开销达到最小，自学了编程然后搞了 K 个数组 color1、color2 … colorK，color1[n]表示**「用 K 种颜色粉刷从第 0 到第 n 个房子所花费的最低开销，其中第 n 个房子粉刷为第 1 种颜色」**。 color2[n] … colorK[n]亦然，粉刷匠每到达一个房子的时候，都会去更新 color1[n]、color2 [n] … colorK[n]，当粉刷匠来到了第 N 个房子的时心里可能这么想：

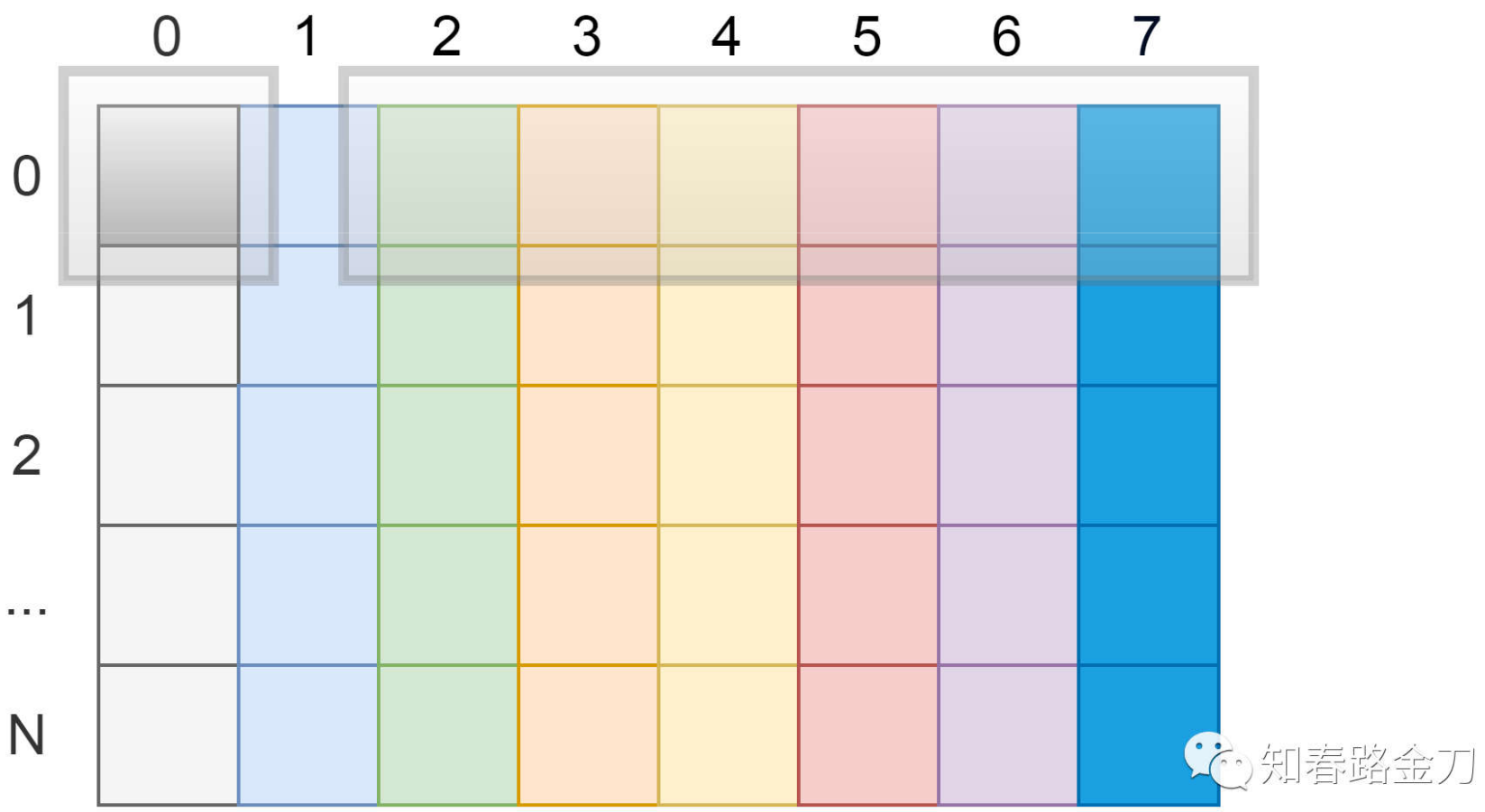
- 我要把第 N 个房子刷为第 1 种颜色
    - 粉刷匠决定把第 N 个房子刷为第 1 种颜色，并记录下当前第 N 个房子刷为第 1 种颜色时所花费的最低开销为 color1[n] = min(color2[n-1], …., colorK[n-1]) + cost[n]。
    - **「解释」**：既然粉刷匠想把第 N 个房子刷为第 1 种颜色，且想保持所花费的开销是最低的，那第 N-1 一个房子不能是第 1 种颜色的且要选择 color2[n-1] …. colorK[n-1]这些开销中最小的一个。
- 我要把第 N 个房子刷为第 2 种颜色
    - 同上
- 我要把第 N 个房子刷为第 K 种颜色
    - 同上

那么我们得出了状态转移方程如下

```
color1[n] = min(color2[n-1], ..., colorK[n-1]) + cost[n]

color2[n] = min(color1[n-1], ..., colorK[n-1]) + cost[n]

...

colorK[n] = min(color1[n-1], ..., colorK-1[n-1]) + cost[n]
```

有了状态转移方程， 那就很容易写出代码了

把 color1,color2 … colorK 变成 dp 数组就变成了如下面这张图的样子



这个例子中，从 0，1 ... 7 共有 8 种颜色，这个 dp 数组中的第 K 列也就代表着 colorK。 这个数组中的第 N 行中的元素表示：**「用 8 种颜色粉刷从第 0 到第 N 个房子，每种颜色所对应的最小开销」**。 例如dp[1][1]的含义为：**「用 8 种颜色粉刷从第 0 到第 1 个房子，且第 1 个房子粉刷成蓝色(也就是第 1 种颜色)时的最小开销」**。

方法一：动态规划

代码实现

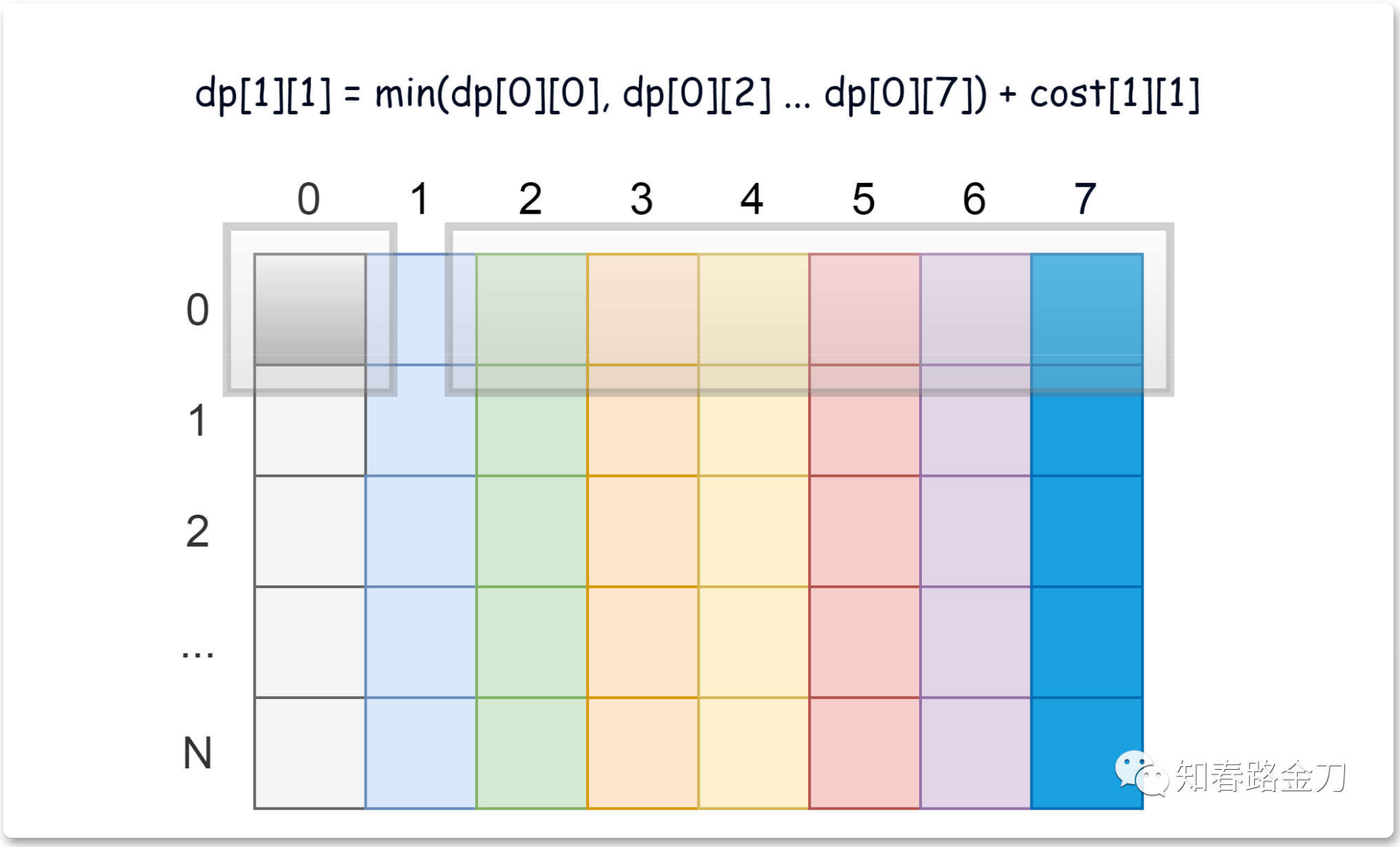
```
class Solution {
    public int minCostII(int[][] costs) {
        int[][] dp = new int[costs.length][costs[0].length];
        int K = costs[0].length;
        for (int k = 0; k < K; k++) {
            dp[0][k] = costs[0][k];
        }
        for (int i = 1; i < costs.length; i++) {
            for (int k = 0; k < K; k++) {
                int min = Integer.MAX_VALUE;
                for (int j = 0; j < K; j++) {
                    if (j != k) {
                        min = Math.min(min, dp[i-1][j]);
                    }
                }
                dp[i][k] = min + costs[i][k];
            }
        }
        int result = Integer.MAX_VALUE;
        for (int i = 0; i < K; i++) {
            result = Math.min(result, dp[dp.length-1][i]);
        }
        return result;
    }
}
```

复杂度分析

N 为房子的数量，K 为颜色的数量

- 时间复杂度: O(N*K2)
- 空间复杂度: O(N*K)

方法二：动态规划优化版



拿上图举例，在方法一中求 dp[1][1] 的值时，最内部的循环只是想计算出除 dp[0][1] 之外的 dp[0][0]到 dp[0][7]之间的最小值。这个 for 循环我们其实可以用两个常量「fristMinCost」和「secondMinCost」来代替，假设我们准备刷第 N 个房子，那么对于这两个变量的解释如下:

- fristMinCost: dp[N-1][0]到 dp[N-1][7]中最小的值(用 8 种颜色刷从第 0 到第 N-1 个房子时的最小开销)。
- secondMinCost: dp[N-1][0]到 dp[N-1][7]中第二小的值(用 8 种颜色刷从第 0 到第 N-1 个房子时的第二小的开销)。

代码实现

```
class Solution {
    public int minCostII(int[][] costs) {
        int[][] dp = new int[costs.length][costs[0].length];
        int fristMinCost = Integer.MAX_VALUE;
        int secondMinCost = Integer.MAX_VALUE;

        int K = costs[0].length;
        for (int k = 0; k < K; k++) {
            dp[0][k] = costs[0][k];
            if (costs[0][k] < fristMinCost) {
                secondMinCost = fristMinCost;
                fristMinCost = costs[0][k];
            } else {
                secondMinCost = Math.min(secondMinCost, costs[0][k]);
            }
        }

        for (int i = 1; i < costs.length; i++) {
            int t1 = Integer.MAX_VALUE, t2 = Integer.MAX_VALUE;
            for (int k = 0; k < K; k++) {
                dp[i][k] = ((dp[i-1][k] == fristMinCost) ? secondMinCost : fristMinCost) + costs[i][k];
                if (dp[i][k] < t1) {
                    t2 = t1;
                    t1 = dp[i][k];
                } else {
                    t2 = Math.min(t2, dp[i][k]);
                }
            }
            fristMinCost = t1;
            secondMinCost = t2;
        }
        int result = Integer.MAX_VALUE;
        for (int i = 0; i < K; i++) {
            result = Math.min(result, dp[dp.length-1][i]);
        }
        return result;
    }
}
```

复杂度分析

N 为房子的数量，K 为颜色的数量

- 时间复杂度: O(N*K)
- 空间复杂度: O(N*K)

其他

> **「图解大厂面试高频算法题」**专题文章主旨是：根据二八法则的原理，以付出 20%的时间成本，获得 80%的刷题的收益，让那些想进互联网大厂或心仪公司的人少走些弯路。
>
> 本专题还在持续更新 ing~ 所有文章、图解和代码全部是金刀亲手完成。内容全部放在了**github**[2]和**gitee**[3]方便小伙伴们阅读和调试，另外还有更多小惊喜等你发现~
>
> 如果你喜欢本篇文章，PLZ 一键三连。

参考资料

[1] **原题链接:**
https://leetcode-cn.com/problems/paint-house-ii/

[2] **github:**
https://github.com/goldknife6

[3] **gitee:**
https://gitee.com/goldknife6