# 题目

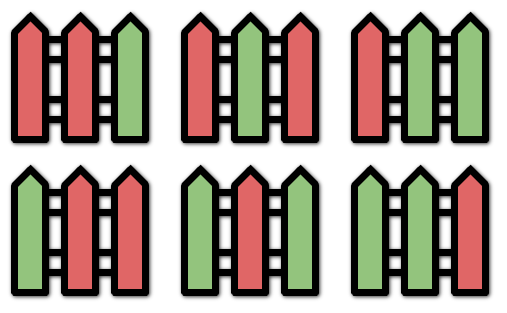
有k种颜色的涂料和一个包含n个栅栏柱的栅栏，请你按下述规则为栅栏设计涂色方案：

每个栅栏柱可以用其中 一种 颜色进行上色。

相邻的栅栏柱 最多连续两个 颜色相同。

给你两个整数k和n ，返回所有有效的涂色 方案数 。

示例 1：



输入：n = 3, k = 2

输出：6

解释：所有的可能涂色方案如上图所示。注意，全涂红或者全涂绿的方案属于无效方案，因为相邻的栅栏柱 最多连续两个 颜色相同。

示例 2：

输入：n = 1, k = 1

输出：1

示例 3：

输入：n = 7, k = 2

输出：42

提示：

1 <= n <= 50

1 <= k <= 105

题目数据保证：对于输入的n和k，其答案在范围[0, 231 - 1]内。

# 分析

## 方法一：动态规划

**思路：**

由于每个栅栏都有两个状态：

0：与上一个颜色相同

1：与上一个颜色不同

那么我们可以开辟一个dp[n][2]大小的数组来表示：

dp[i][0] // 状态 0 表示与上一个颜色相同

dp[i][1] // 状态 1 表示与上一个颜色不同

状态转移方程也很容易写出来：

dp[i][0] = dp[i - 1][1];

dp[i][1] = (dp[i - 1][0] + dp[i - 1][1]) \* (k - 1);

最结果为dp[n - 1][0] + dp[n - 1][1]

**代码：**

class Solution {

public:

int numWays(int n, int k) {

if (k == 1) return n > 2 ? 0 : 1;

if (n <= 2) return pow(k, n);

vector<vector<int>> dp(n, vector<int>(2));

dp[0][0] = 0; // 状态 0 表示与上一个颜色相同

dp[0][1] = k; // 状态 1 表示与上一个颜色不同

for (int i = 1; i < n; ++i) {

dp[i][0] = dp[i - 1][1];

dp[i][1] = (dp[i - 1][0] + dp[i - 1][1]) \* (k - 1);

}

return dp[n - 1][0] + dp[n - 1][1];

}

};

另一种写法：

class Solution {

public:

int numWays(int n, int k) {

if (!n) {

return 0;

}

if (n == 1) {

return k;

}

int diff = k \* (k - 1);

int same = k;

for (int i = 3; i <= n; ++i) {

int tmp = diff;

diff = same \* (k - 1) + diff \* (k - 1);

same = tmp;

}

return diff + same;

}

};

## 方法二：一维dp

由于每个栅栏都有两个状态：

0：与上一个颜色相同

1：与上一个颜色不同

开辟一个dp[n]大小的数组来表示n有效的方案数：

对于状态1，与上一个颜色不同，dp[i] = dp[i - 1] \* (k - 1); 这个好理解。

对于状态 0，与上一个颜色相同，我们可以把当前 i 和其上一个 i - 1 看做一个大栅栏，这个大栅栏不能与 i - 2 的栅栏颜色相同，于是又回到了状态 1，两个栅栏的颜色不同，那么状态转移方程也能轻松写出来：

dp[i] = dp[i - 2] \* (k - 1);

最终的转移方程为 dp[i] = dp[i - 1] \* (k - 1) + dp[i - 2] \* (k - 1);

代码：

class Solution {

public:

int numWays(int n, int k) {

if (k == 1) return n > 2 ? 0 : 1;

if (n <= 2) return pow(k, n);

vector<int> dp(n);

dp[0] = k;

dp[1] = k \* k;

for (int i = 2; i < n; ++i) {

dp[i] = dp[i - 1] \* (k - 1) + dp[i - 2] \* (k - 1);

}

return dp[n - 1];

}

};