<https://leetcode.ca/all/276.html>

There is a fence with n posts, each post can be painted with one of the k colors.

You have to paint all the posts such that no more than two adjacent fence posts have the same color.

Return the total number of ways you can paint the fence.

**Note:** n and k are non-negative integers.

**Example:**

Input: n = 3, k = 2

Output: 6

Explanation: Take c1 as color 1, c2 as color 2. All possible ways are:

post1 post2 post3

----- ----- ----- -----

1 c1 c1 c2

2 c1 c2 c1

3 c1 c2 c2

4 c2 c1 c1

5 c2 c1 c2

6 c2 c2 c1

**Attempt 1: 2023-11-26**

**Solution 1: Native DFS (120 min)**

**Style 1: 顶{index=0,sameColor=true}和{index=0,sameColor=false}，底{index=n-1,sameColor=true}和{index=n-1,sameColor=false}**

class Solution {

public int numWays(int n, int k) {

// Boolean flag 'sameColor' means repeating the same color

// on the last two posts or not

return helper(0, true, n, k) + helper(0, false, n, k);

}

private int helper(int index, boolean sameColor, int n, int k) {

if(index == n) {

return 0;

}

// Base condition

// We cannot simply write as when index == n - 1 -> return k, the additional mandatory

// condition is 'sameColor' flag is 'false', since when index == n - 1, 'sameColor'

// flag is 'true' not applicable, no previous post to consider

// Map to below dp statement:

// There are k ways to paint the first post (since it has no previous post to consider)

// dp[n - 1][0] = k;

// dp[n - 1][1] is not applicable because there is no previous post

if(index == n - 1 && !sameColor) {

return k;

}

int count = 0;

if(sameColor) {

count += helper(index + 1, false, n, k);

} else {

count += (helper(index + 1, true, n, k) + helper(index + 1, false, n, k)) \* (k - 1);

}

return count;

}

}

Time Complexity: O(2^N)

Space Complexity: O(2^N)

**Style 2: 顶{index=n-1,sameColor=true}和{index=n-1,sameColor=false}，底{index=0,sameColor=true}和{index=0,sameColor=false}**

class Solution {

public int numWays(int n, int k) {

// Boolean flag 'sameColor' means repeating the same color

// on the last two posts or not

return helper(n - 1, true, k) + helper(n - 1, false, k);

}

private int helper(int index, boolean sameColor, int k) {

if(index < 0) {

return 0;

}

// Base condition

// We cannot simply write as when index == 0 -> return k, the additional mandatory

// condition is 'sameColor' flag is 'false', since when index == 0, 'sameColor'

// flag is 'true' not applicable, no previous post to consider

// Map to below dp statement:

// There are k ways to paint the first post (since it has no previous post to consider)

// dp[0][0] = k;

// dp[0][1] is not applicable because there is no previous post

if(index == 0 && !sameColor) {

return k;

}

int count = 0;

if(sameColor) {

count += helper(index - 1, false, k);

} else {

count += (helper(index - 1, true, k) + helper(index - 1, false, k)) \* (k - 1);

}

return count;

}

}

Time Complexity: O(2^N)

Space Complexity: O(2^N)

**Step by Step for Style 2:**

(k=2 is fixed, initial n=3)

return 4+2=6

/ \

/ \

Start branch 1 Start branch 2

helper(n-1,false,k) helper(n-1,true,k)

index=2 index=2

sameColor=false sameColor=true

return (2+2)\*(2-1)=4 return 2

/ \ |

/ \ |

index=1 index=1 index=1

sameColor=false sameColor=true sameColor=false

return (2+0)\*(2-1)=2 return 2 return (2+0)\*(2-1)=2

/ \ | / \

/ \ | / \

index=0 index=0 index=0 index=0 index=0

sameColor=false true sameColor=false sameColor=false true

return k=2 return 0 return k=2 return k=2 return 0

**Solution 2: DFS + Memoization (10 min)**

**Style 1: 基于Solution 1 Style 1的顶{index=0,sameColor=true}和{index=0,sameColor=false}，底{index=n-1,sameColor=true}和{index=n-1,sameColor=false}**

class Solution {

public int numWays(int n, int k) {

// Additional '+1' -> for base condition: index == n

Integer[][] memo = new Integer[n + 1][k];

// Boolean flag 'sameColor' means repeating the same color

// on the last two posts or not

return helper(0, 1, n, k, memo) + helper(0, 0, n, k, memo);

}

private int helper(int index, int sameColor, int n, int k, Integer[][] memo) {

if(index == n) {

return memo[index][sameColor] = 0;

}

// Base condition

// We cannot simply write as when index == n - 1 -> return k, the additional mandatory

// condition is 'sameColor' flag is 'false(=0)', since when index == n - 1, 'sameColor'

// flag is 'true(=1)' not applicable, no previous post to consider

// Map to below dp statement:

// There are k ways to paint the first post (since it has no previous post to consider)

// dp[n - 1][0] = k;

// dp[n - 1][1] is not applicable because there is no previous post

if(index == n - 1 && sameColor == 0) {

return memo[index][sameColor] = k;

}

if(memo[index][sameColor] != null) {

return memo[index][sameColor];

}

int count = 0;

if(sameColor == 1) {

count += helper(index + 1, 0, n, k, memo);

} else {

count += (helper(index + 1, 1, n, k, memo) + helper(index + 1, 0, n, k, memo)) \* (k - 1);

}

return memo[index][sameColor] = count;

}

}

**Style 2: 基于Solution 1 Style 1的顶{index=n-1,sameColor=true}和{index=n-1,sameColor=false}，底{index=0,sameColor=true}和{index=0,sameColor=false}**

class Solution {

public int numWays(int n, int k) {

// No need '+1' because base condition 'index < 0' no need '+1' to handle

Integer[][] memo = new Integer[n][k];

// Boolean flag 'sameColor' means repeating the same color

// on the last two posts or not

return helper(n - 1, 1, k, memo) + helper(n - 1, 0, k, memo);

}

private int helper(int index, int sameColor, int k, Integer[][] memo) {

if(index < 0) {

// Cannot cover negative index, directly return 0

//return memo[index][sameColor] = 0;

return 0;

}

// Base condition

// We cannot simply write as when index == 0 -> return k, the additional mandatory

// condition is 'sameColor' flag is 'false(=0)', since when index == 0, 'sameColor'

// flag is 'true(=1)' not applicable, no previous post to consider

// Map to below dp statement:

// There are k ways to paint the first post (since it has no previous post to consider)

// dp[n - 1][0] = k;

// dp[n - 1][1] is not applicable because there is no previous post

if(index == 0 && sameColor == 0) {

return memo[index][sameColor] = k;

}

if(memo[index][sameColor] != null) {

return memo[index][sameColor];

}

int count = 0;

if(sameColor == 1) {

count += helper(index - 1, 0, k, memo);

} else {

count += (helper(index - 1, 1, k, memo) + helper(index - 1, 0, k, memo)) \* (k - 1);

}

return memo[index][sameColor] = count;

}

}

**Solution 3: DP + Fibonacci (120 min)**

**Style 1: 对应Native DFS中Style 1顶{index=0,sameColor=true}和{index=0,sameColor=false}，底{index=n-1,sameColor=true}和{index=n-1,sameColor=false}，DP顶dp[0][0]和dp[0][1]，底dp[n-1][0]和dp[n-1][1]**

class Solution {

public int numWays(int n, int k) {

// dp[i][0] represents the number of ways to paint the fence up to post i

// without repeating colors on the last two posts

// dp[i][1] represents the number of ways to paint the fence up to post i

// with the same color on the last two posts

int[][] dp = new int[n][2];

// Base condition

// There are k ways to paint the first post (since it has no previous post to consider)

dp[n - 1][0] = k;

// dp[0][1] is not applicable because there is no previous post

for(int i = n - 2; i >= 0; i--) {

// Paint the current post without repeating colors

// The last two posts can be either different or the same, but the current

// post must be a different color than the last one, hence (k - 1) choices

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

// Paint the current post using the same color

// The last two posts must be of different colors for the current post to

// have the same color as the last one. There is exactly one way to paint it,

// which is the same color as the previous post.

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

}

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

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Style 2: 对应Native DFS中Style 2顶{index=n-1,sameColor=true}和{index=n-1,sameColor=false}，底{index=0,sameColor=true}和{index=0,sameColor=false}，DP顶dp[n-1][0]和dp[n-1][1]，底dp[0][0]和dp[0][1]**

class Solution {

public int numWays(int n, int k) {

// dp[i][0] represents the number of ways to paint the fence up to post i

// without repeating colors on the last two posts

// dp[i][1] represents the number of ways to paint the fence up to post i

// with the same color on the last two posts

int[][] dp = new int[n][2];

// Base condition

// There are k ways to paint the first post (since it has no previous post to consider)

dp[0][0] = k;

// dp[0][1] is not applicable because there is no previous post

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

// Paint the current post without repeating colors

// The last two posts can be either different or the same, but the current

// post must be a different color than the last one, hence (k - 1) choices

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

// Paint the current post using the same color

// The last two posts must be of different colors for the current post to

// have the same color as the last one. There is exactly one way to paint it,

// which is the same color as the previous post.

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

}

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

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://algo.monster/liteproblems/276>

## Intuition

The solution to this problem can be approached using dynamic programming because the way you paint the current post is dependent on how you painted the previous ones. The key insight is that the number of ways to paint the last post depends on the color of the second to last post.

Let's define two scenarios:

* The last two posts have the same color.
* The last two posts have different colors.

Given that we cannot paint three consecutive posts with the same color, if the last two posts are of the same color, the current post can only be painted with any of the k-1 remaining colors. If the last two posts have different colors, then the current post can also be painted with any of the k-1 remaining colors. However, network need to consider different ways to reach these scenarios.

Thus, we define a dynamic programming table dp where dp[i][0] represents the number of ways to paint up to post i where the last two posts are of different colors, and dp[i][1] represents the number of ways where the last two posts are the same.

The recurrence relations based on the defined scenarios are as follows:

* dp[i][0] (different colors) = (dp[i - 1][0] + dp[i - 1][1]) \* (k - 1)
* dp[i][1] (same colors) = dp[i - 1][0]

The reason behind these relations is that, for dp[i][0], you have both previous scenarios possible and you cannot use the last color used, hence k - 1 options. For dp[i][1], you can only come from the scenario where previous colors are different, and you must use the same color as the last one, hence only one option available.

The initial conditions are:

* dp[0][0] = k (because the first post can be any of the k colors)
* dp[0][1] is not applicable because there is no previous post

We iterate through the posts, calculating the number of ways based on the above recurrence relations. Finally, the answer is the sum of ways for the last post being either the same or different colors as the one before it, i.e., sum(dp[-1]).

## Solution Approach

The given solution implements the intuition using a dynamic programming (DP) approach. The essential components of this solution are:

* **Data Structure:** A 2D list dp with n rows and 2 columns, used to store the number of ways to paint up to the current post. Here, n is the number of fence posts. The first column (dp[i][0]) stores the number of ways if the last two posts have different colors, and the second column (dp[i][1]) stores the number if they are the same.
* **Initialization:** The DP table is initialized with dp[0][0] = k, representing the k ways to paint the first fence. Since there is no post before the first one, we don't need to initialize dp[0][1] as it is not applicable.
* **Iteration:** The code iterates over each post from 1 to n-1. For each post, the following calculations are performed:

1. dp[i][0] = (dp[i - 1][0] + dp[i - 1][1]) \* (k - 1): The last two posts can be either different or the same, but the current post must be a different color than the last one, hence (k - 1) choices.
2. dp[i][1] = dp[i - 1][0]: The last two posts must be of different colors for the current post to have the same color as the last one. There is exactly one way to paint it, which is the same color as the previous post.

* **Final Calculation:** After filling the DP table up to n posts, the total ways to paint the fence can be found by adding the values of the last row of the DP table: sum(dp[-1]). This is because the final post can either be painted the same or a different color from the one before it, and we are interested in all possible valid combinations.

The core of this algorithm lies in understanding the constraints and how they impact the subsequences and the transitions between states in the dynamic programming table. This solution maintains constant space complexity for each post with respect to the number of colors, making the overall space complexity O(n). The time complexity is O(n) as well because we compute the entries of the DP table with constant time operations for each of the n posts.

The final result is returned by summing the two scenarios in the last row, which provides the count of all the ways to paint the fence according to the rules.

### Example Walkthrough

Let's walk through an example to illustrate the solution approach using the given problem and intuition. Suppose we have n = 3 posts to paint and k = 2 different colors. We want to find out how many ways we can paint the fence.

We initialize a DP table dp with dimensions [n][2] where n is the number of fence posts. Each entry dp[i][0] will store ways we can paint up to the i-th post with the last two posts having different colors, and dp[i][1] will store ways with the last two posts having the same color.

**Step 1: Initialization**

For the first fence post (i = 0), we have k options, assuming k is not zero. So, dp[0][0] = k since there is no previous post, and the condition for dp[0][1] is not applicable.

For k = 2, the initialization would be dp[0][0] = 2 (we have 2 ways to paint the first post as there are 2 colors).

**Step 2: Iteration for the second post**

Now, let's move to the second post i = 1. We have two scenarios:

* dp[1][0] = (dp[0][0] + dp[0][1]) \* (k - 1) since we can paint the second post with a different color than the first in k - 1 ways. Here, dp[0][1] can be considered 0 because it's not applicable. So, dp[1][0] = (2 + 0) \* (2 - 1) = 2.
* dp[1][1] = dp[0][0] since the last two posts can be the same if the first post was unique, which is already counted in dp[0][0]. Therefore, dp[1][1] = 2.

At this point, our DP table for i = 1 looks like this:

1dp = [

2 [2, X], // X denotes non-applicable

3 [2, 2],

4]

**Step 3: Iteration for the third post**

For the third post i = 2, we follow a similar procedure:

* dp[2][0] = (dp[1][0] + dp[1][1]) \* (k - 1) which translates to (2 + 2) \* (2 - 1) = 4. We have 4 ways to paint the third post with a different color than the second post.
* dp[2][1] = dp[1][0] since again, the last two can be the same only if the previous two were different. So we take the value from dp[1][0] which is 2, giving us dp[2][1] = 2.

Now, our DP table for i = 2 looks like this:

1dp = [

2 [2, X],

3 [2, 2],

4 [4, 2],

5]

**Step 4: Final Calculation**

After computing all the values, we want the sum of the last row to get the total number of ways to paint the fence according to the problem's rules. sum(dp[2]) = dp[2][0] + dp[2][1] = 4 + 2 = 6.

Thus, there are 6 different ways to paint the 3 posts using 2 colors while following the given rules. This completes the example walk-through using the dynamic programming solution approach described in the content.

**Code**

class Solution {

public int numWays(int n, int k) {

// Create a 2D array to use as a dynamic programming table

// dp[i][0] represents the number of ways to paint the fence up to post i

// without repeating colors on the last two posts

// dp[i][1] represents the number of ways to paint the fence up to post i

// with the same color on the last two posts

int[][] dp = new int[n][2];

// Base case initialization:

// There are k ways to paint the first post (since it has no previous post to consider)

dp[0][0] = k;

// Iterate over the fence posts starting from the second post

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

// Calculate the number of ways to paint the current post without repeating colors

// This is done by multiplying the total number of ways to paint the previous post

// by (k - 1), since we can choose any color except the one used on the last post

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

// Calculate the number of ways to paint the current post using the same color

// as the last post. This can only be done if the last two posts have different colors,

// so we use the value from dp[i - 1][0].

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

}

// Return the total number of ways to paint the entire fence with n posts by summing

// the ways to paint with the same color and with different colors on the last two posts

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

}

}

**Solution 4: DP + Math (30 min)**

class Solution {

public int numWays(int n, int k) {

// dp[0], dp[3] both initialized as 0

int[] dp = new int[4];

// 第一根涂色的方式有k中，第二根涂色的方式则是k\*k，因为第二根柱子可以和第一根一样

dp[1] = k;

dp[2] = k \* k;

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

// 递推式：第三根柱子要么根第一个柱子不是一个颜色，要么跟第二根柱子不是一个颜色

dp[3] = (dp[1] + dp[2]) \* (k - 1);

dp[1] = dp[2];

dp[2] = dp[3];

}

return dp[3];

}

}

Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

<https://segmentfault.com/a/1190000003790650>

#### 复杂度

时间 O(N) 空间 O(1)

#### 思路

这种给定一个规则，计算有多少种结果的题目一般都是动态规划，因为我们可以从这个规则中得到递推式。根据题意，不能有超过连续两根柱子是一个颜色，也就意味着第三根柱子要么根第一个柱子不是一个颜色，要么跟第二根柱子不是一个颜色。如果不是同一个颜色，计算可能性的时候就要去掉之前的颜色，也就是k-1种可能性。假设dp[1]是第一根柱子及之前涂色的可能性数量，dp[2]是第二根柱子及之前涂色的可能性数量，则dp[3]=(k-1)\*dp[1] + (k-1)\*dp[2]。

递推式有了，下面再讨论下base情况，所有柱子中第一根涂色的方式有k中，第二根涂色的方式则是k\*k，因为第二根柱子可以和第一根一样。

public class Solution {

public int numWays(int n, int k) {

// 当n=0时返回0

int dp[] = {0, k , k\*k, 0};

if(n <= 2){

return dp[n];

}

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

// 递推式：第三根柱子要么根第一个柱子不是一个颜色，要么跟第二根柱子不是一个颜色

dp[3] = (k - 1) \* (dp[1] + dp[2]);

dp[1] = dp[2];

dp[2] = dp[3];

}

return dp[3];

}

}

或者直接把dp数组变成变量

**Refer to**

<https://grandyang.com/leetcode/276/>

class Solution {

public:

int numWays(int n, int k) {

if (n == 0) return 0;

int same = 0, diff = k;

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

int t = diff;

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

same = t;

}

return same + diff;

}

};

**Also** **refer to**

<https://medium.com/@xiaogegexiao/interesting-paint-fence-algorithm-36e7e2ab4111>

So the very straightforward method is using DFS(Depth first search). But I thought this issue should be some mathematical issue which can be solved by ***permutation and combination*** , So I decide to come up with a mathematical formula which can be used to scale down the f(n) function to be calculated by smaller input solution such as f(n-1), f(n-2), f(n-3)….

# Mathematical way

Okay, let’s start the formula derivation. Let’s define function f(n) is for calculating how many ways to paint a fence with n points and k colors. So here I want to figure out the relationship between function f(n) for n points fence and less points fence solutions like f(n-1), f(n-2), f(n-3);

***f(n) = f(n-1)\*k***

Here the first thought would be f(n) is actually f(n-1) multiply k colors ways. But what could happen is that there may exists some invalid paint ways if ***the last 1st color is equivalent to the last 2nd and last 3rd color.*** Okay, then to me the way to avoid these invalid ways is just subtract those invalid ways from f(n-1).k. So the only invalid ways for paint fence with appending the last color is f(n-3) \* (k-1). That’s because it will only be invalid if the last 3 colors are the same. Let’s assume that both f(n-3) and f(n -2) are valid result, which means the last 3rd color is mush not equivalent to the last 4th color. And then the last 3 colors have (k-1) different choices

Then the new formula will be

***f(n) = f(n-1)\*k- f(n-3)\*(k-1)***

Cool, after the mathematical formula derived. Translating it to code is quite easy. Here we use very simple recursive and dynamic programming and Kotlin as language. The edge cases here to be considered should be in sequence(n == 1, n == 2, k < 2 and n == 3). After n > 3, it will do recursive calculation

/\*\*

\* @param n: non-negative integer, n posts

\* @param k: non-negative integer, k colors

\* @return: an integer, the total number of ways

\*/

fun numWays(n: Int, k: Int): Int {

if (n == 1) {

return k

}

if (n == 2) {

return k \* k

}

if (k < 2) {

return 0

}

return if (n == 3) {

k \* (k - 1 + (k - 1) \* k)

} else {

k \* numWays(n - 1, k) - numWays(n - 3, k) \* (k - 1)

}

}

Also, if people still get interested in DFS (Depth first search) way. I also put down the code here. DFS way is just basically try to iterate all the possible path for painting the fence.

# Depth first search way (Backtracking)

fun numWays(n: Int, k: Int): Int {

if (n == 1) {

return k

}

if (n == 2) {

return k \* k

}

if (k < 2) {

return 0

}

val fence = IntArray(n)

Arrays.fill(fence, -1)

val result = IntArray(1)

dfs(result, fence, 0, n, k)

return result[0]

}

private fun dfs(result: IntArray, fence: IntArray, index: Int, n: Int, k: Int) {

if (index == n) {

result[0]++

return

}

for (i in 0 until k) {

if (index >= 2 && fence[index - 1] == fence[index - 2] && i == fence[index - 1]) {

continue

}

fence[index] = i

dfs(result, fence, index + 1, n, k)

fence[index] = -1

}

}