1931. Painting a Grid With Three Different Colors

Leetcode Link

Problem Description

Given two integers `m` and `n`, consider an `m x n` grid where each cell is initially white. You can paint each cell one of three colors: red, green, or blue. All cells must be painted. The task is to find the number of ways to color the grid such that no two adjacent cells (horizontally or vertically) have the same color. Since the answer can be very large, return it modulo 10^9 + 7.

Example

Suppose you are given `m = 2` and `n = 2`, the grid would look like:

```
1
2
3 _ _
4 _ _
```

There are 6 ways to color the grid with no two adjacent cells having the same color:

```
 1
 2
 3 1. R G   2. R B   3. G R   4. G B   5. B R   6. B G
 4    G R      G B      B R      B G      G B      R G
```

So, the answer for this example would be 6.

Approach

To solve the problem, we will use dynamic programming. The following steps will be performed in the DP-based solution:

1. Create a recursive function `dp` with four parameters: the row number `r`, the column number `c`, the mask representing the previous column `prevColMask`, and the mask representing the current column `currColMask`.
2. The base case is when the column number `c` is equal to `n`. In this case, return 1 as there's only one way left to color the last cell.
3. Check if the result for the current column is already memoized in the `memo` array. If it is, return the memoized value.
4. When row `r` is equal to `m`, we have reached the end of the current column. Call `dp` recursively for the next column, updating the `prevColMask` and `currColMask` accordingly.
5. Iterate through the colors (1 for red, 2 for green, 3 for blue) and check if the current color can be used to paint the cell at position `(r, c)`. If the color is the same as the one in the cell above or the one to the left, skip this color.
6. Call the `dp` function recursively using the selected color, and update the `currColMask` accordingly.
7. Store the computed result in the `memo` array for future use, and return it.

To facilitate mask operations, we will use helper functions `getColor` and `setColor`.

Example Walkthrough

Suppose we have a 1×3 grid, then m = 1 and n = 3.

1. Initialize the `memo` array as a 1000×1024 vector of zeroes.
2. Call recursively the `dp` function with the initial values:

```
1
2
3 m = 1, n = 3
4 dp(0, 0, 0, 0)
```

After calling the `dp` function, the algorithm will go through steps 1-7 described in the Approach section. The output value will be 6, which is the number of ways to color the 1×3 grid.

Solution in Python

Here's the solution in Python:

```
 1
 2 python
 3 class Solution:
 4     kMod = 1000000007
 5
 6     def colorTheGrid(self, m: int, n: int) -> int:
 7         self.m = m
 8         self.n = n
 9         self.memo = [[0] * 1024 for _ in range(1000)]
10         return self.dp(0, 0, 0, 0)
11
12     def dp(self, r: int, c: int, prevColMask: int, currColMask: int) -> int:
13         if c == self.n:
14             return 1
15         if self.memo[c][prevColMask]:
16             return self.memo[c][prevColMask]
17         if r == self.m:
18             return self.dp(0, c + 1, currColMask, 0)
19
20         ans = 0
21         for color in range(1, 4):
22             if self.getColor(prevColMask, r) == color: continue
23             if r > 0 and self.getColor(currColMask, r - 1) == color: continue
24             ans += self.dp(r + 1, c, prevColMask, self.setColor(currColMask, r, color))
25             ans %= self.kMod
26
27         if r == 0:
28             self.memo[c][prevColMask] = ans
29
30         return ans
31
32     def getColor(self, mask: int, r: int) -> int:
33         return (mask >> (r * 2)) & 3
34
35     def setColor(self, mask: int, r: int, color: int) -> int:
36         return mask | (color << (r * 2))
```

This solution follows the approach we explained before, with appropriate syntax for Python.## Solution in JavaScript

Here's the solution in JavaScript:

```
 1
 2 javascript
 3 class Solution {
 4     kMod = 1000000007;
 5
 6     colorTheGrid(m, n) {
 7         this.m = m;
 8         this.n = n;
 9         this.memo = Array.from({ length: 1000 }, () => Array(1024).fill(0));
10         return this.dp(0, 0, 0, 0);
11     }
12
13     dp(r, c, prevColMask, currColMask) {
14         if (c === this.n) {
15             return 1;
16         }
17         if (this.memo[c][prevColMask]) {
18             return this.memo[c][prevColMask];
19         }
20         if (r === this.m) {
21             return this.dp(0, c + 1, currColMask, 0);
22         }
23
24         let ans = 0;
25         for (let color = 1; color <= 3; color++) {
26             if (this.getColor(prevColMask, r) === color) continue;
27             if (r > 0 && this.getColor(currColMask, r - 1) === color) continue;
28             ans += this.dp(r + 1, c, prevColMask, this.setColor(currColMask, r, color));
29             ans %= this.kMod;
30         }
31
32         if (r === 0) {
33             this.memo[c][prevColMask] = ans;
34         }
35
36         return ans;
37     }
38
39     getColor(mask, r) {
40         return (mask >> (r * 2)) & 3;
41     }
42
43     setColor(mask, r, color) {
44         return mask | (color << (r * 2));
45     }
46 }
```

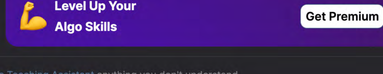
This solution follows the same approach we explained before, with appropriate syntax for JavaScript.

Solution in Java

Here's the solution in Java:

```
 1
 2 java
 3 class Solution {
 4     private static int kMod = 1000000007;
 5
 6     private int m;
 7     private int n;
 8     private int[][] memo;
 9
10     public int colorTheGrid(int m, int n) {
11         this.m = m;
12         this.n = n;
13         this.memo = new int[1000][1024];
14         return dp(0, 0, 0, 0);
15     }
16
17     private int dp(int r, int c, int prevColMask, int currColMask) {
18         if (c == n) {
19             return 1;
20         }
21         if (memo[c][prevColMask] != 0) {
22             return memo[c][prevColMask];
23         }
24         if (r == m) {
25             return dp(0, c + 1, currColMask, 0);
26         }
27
28         int ans = 0;
29         for (int color = 1; color <= 3; color++) {
30             if (getColor(prevColMask, r) == color) continue;
31             if (r > 0 && getColor(currColMask, r - 1) == color) continue;
32             ans += dp(r + 1, c, prevColMask, setColor(currColMask, r, color));
33             ans %= kMod;
34         }
35
36         if (r == 0) {
37             memo[c][prevColMask] = ans;
38         }
39
40         return ans;
41     }
42
43     private int getColor(int mask, int r) {
44         return (mask >> (r * 2)) & 3;
45     }
46
47     private int setColor(int mask, int r, int color) {
48         return mask | (color << (r * 2));
49     }
50 }
```

This solution follows the approach we explained before, with appropriate syntax for Java.



Got a question? Ask the Teaching Assistant anything you don't understand.