Paint Fence

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.

Solution 1

```java
public int numWays(int n, int k) {
    if(n == 0) return 0;
    else if(n == 1) return k;
    int diffColorCounts = k*(k-1);
    int sameColorCounts = k;
    for(int i=2; i<n; i++) {
        int temp = diffColorCounts;
        diffColorCounts = (diffColorCounts + sameColorCounts) * (k-1);
        sameColorCounts = temp;
    }
    return diffColorCounts + sameColorCounts;
}
```

We divided it into two cases.

1. the last two posts have the same color, the number of ways to paint in this case is *sameColorCounts*.
2. the last two posts have different colors, and the number of ways in this case is *diffColorCounts*.

The reason why we have these two cases is that we can easily compute both of them, and that is all I do. When adding a new post, we can use the same color as the last one (if allowed) or different color. If we use different color, there're *k-1* options, and the outcomes shoule belong to the *diffColorCounts* category. If we use same color, there's only one option, and we can only do this when the last two have different colors (which is the diffColorCounts). There we have our induction step.

Here is an example, let's say we have 3 posts and 3 colors. The first two posts we have 9 ways to do them, (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3). Now we know that

```
diffColorCounts = 6;
```

And

```
sameColorCounts = 3;
```

Now for the third post, we can compute these two variables like this:

If we use different colors than the last one (the second one), these ways can be added into *diffColorCounts*, so if the last one is 3, we can use 1 or 2, if it's 1, we can use 2 or 3, etc. Apparently there are `(diffColorCounts + sameColorCounts) * (k-1)` possible ways.

If we use the same color as the last one, we would trigger a violation in these three cases (1,1,1), (2,2,2) and (3,3,3). This is because they already used the same color for the last two posts. So is there a count that rules out these kind of cases? YES, the *diffColorCounts*. So in cases within *diffColorCounts*, we can use the same color as the last one without worrying about triggering the violation. And now as we append a same-color post to them, the former *diffColorCounts* becomes the current *sameColorCounts*.

Then we can keep going until we reach the n. And finally just sum up these two variables as result.

Hope this would be clearer.

written by JennyShaw original link here

Solution 2

If n == 1, there would be k-ways to paint.

if n == 2, there would be two situations:

- 2.1 You paint same color with the previous post: k*1 ways to paint, named it as `same`
- 2.2 You paint differently with the previous post: k*(k-1) ways to paint this way, named it as `dif`

So, you can think, if n >= 3, you can always maintain these two situations, `You either paint the same color with the previous one, or differently`.

Since there is a rule: "no more than two adjacent fence posts have the same color."

We can further analyze:

- from 2.1, since previous two are in the same color, next one you could only paint differently, and it would form one part of "paint differently" case in the n == 3 level, and the number of ways to paint this way would equal to `same*(k-1)`.
- from 2.2, since previous two are not the same, you can either paint the same color this time (`dif*1`) ways to do so, or stick to paint differently (`dif*(k-1)`) times.

Here you can conclude, when seeing back from the next level, ways to paint the same, or variable `same` would equal to `dif*1 = dif`, and ways to paint differently, variable `dif`, would equal to `same*(k-1)+dif*(k-1) = (same + dif)*(k-1)`

So we could write the following codes:

```
    if n == 0:
        return 0
    if n == 1:
        return k
    same, dif = k, k*(k-1)
    for i in range(3, n+1):
        same, dif = dif, (same+dif)*(k-1)
    return same + dif
```

written by chungyushao original link here

Solution 3

Need two one-dimensional array dp1 and dp2, dp1[i] means the number of solutions when the color of last two fences (whose indexes are i-1,i-2) are same. dp2[i] means the number of solutions when the color of last two fences are different.

So

dp1[i]=dp2[i-1],

dp2[i]=(k-1)*(dp1[i-1]+dp2[i-1]) =(k-1)*(dp2[i-2]+dp2[i-1])

Final result is dp1[n-1]+dp2[n-1];

In the code, variable *a,c* mean the last two items of dp1, variable*b,d* mean the last two items of dp2, and c could be eliminated.

```
class Solution {
public:
    int numWays(int n, int k) {
        if(n<=1 || k==0)return n*k;
        int a=k,b=k*(k-1),c=0,d=0;
        for(int i=2;i<n;++i){
            d=(k-1)*(a+b);
            a=b;b=d;
        }
        return a+b;
    }
};
```

written by jiannan original link here