> PS：动态规划题型中最最经典的入门题目。

题目介绍[1]

给你一个整数数组 nums ，请你找出数组中乘积最大的连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。

示例 1

> 输入：[2,3,-2,4]
>
> 输出：6
>
> 解释：子数组 [2,3] 有最大乘积 6。

示例 2

> 输入：[-2,0,-1]
>
> 输出：0
>
> 解释：结果不能为 2，因为 [-2,-1] 不是子数组。

题目解答

方法一：一维动态规划

思路和算法

子数组构成的要素有这么一个关键点：这个数组肯定有一个起始元素和一个末尾元素。如果我们算出了以 **「nums[i-1]为结尾的子数组乘积的最大值」**，那以 **「nums[i]为结尾的子数组乘积的最大值」** 是不是就是 **「max(nums[i-1]为结尾的子数组乘积的最大值 * nums[i]，nums[i])」** ？

大家可以仔细思考一下，其实离正确答案很接近了，就差了一点点。因为没有考虑到 nums 数组里有负数的情况。nums[i]为负数的时候，如果可以找到一个以 **「nums[i-1] 为结尾的子数组乘积的最小值(最好也是负数)」**，与 nums[i]相乘是不是就有可能得到一个更大的数？确实如此，那我们尝试去写一下状态转移方程。

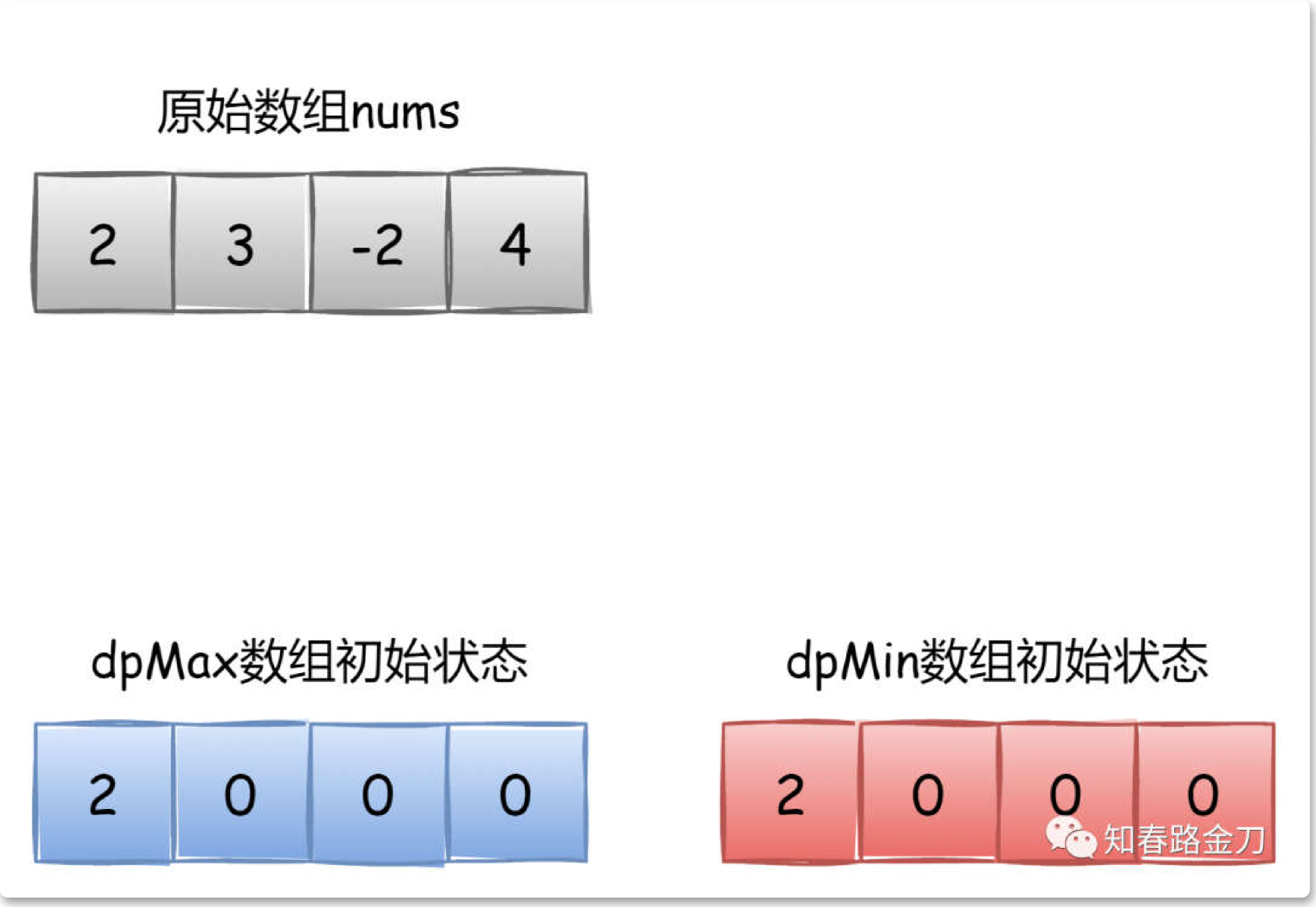
构建两个 dp 数组dpMax和dpMin

> dpMax[i] 表示以 nums[i]元素结尾的子数组的乘积，且乘积是最大的。
>
> dpMin[i] 表示以 nums[i]元素结尾的子数组的乘积，且乘积是最小的。

这样我们就可以得出状态转移方程

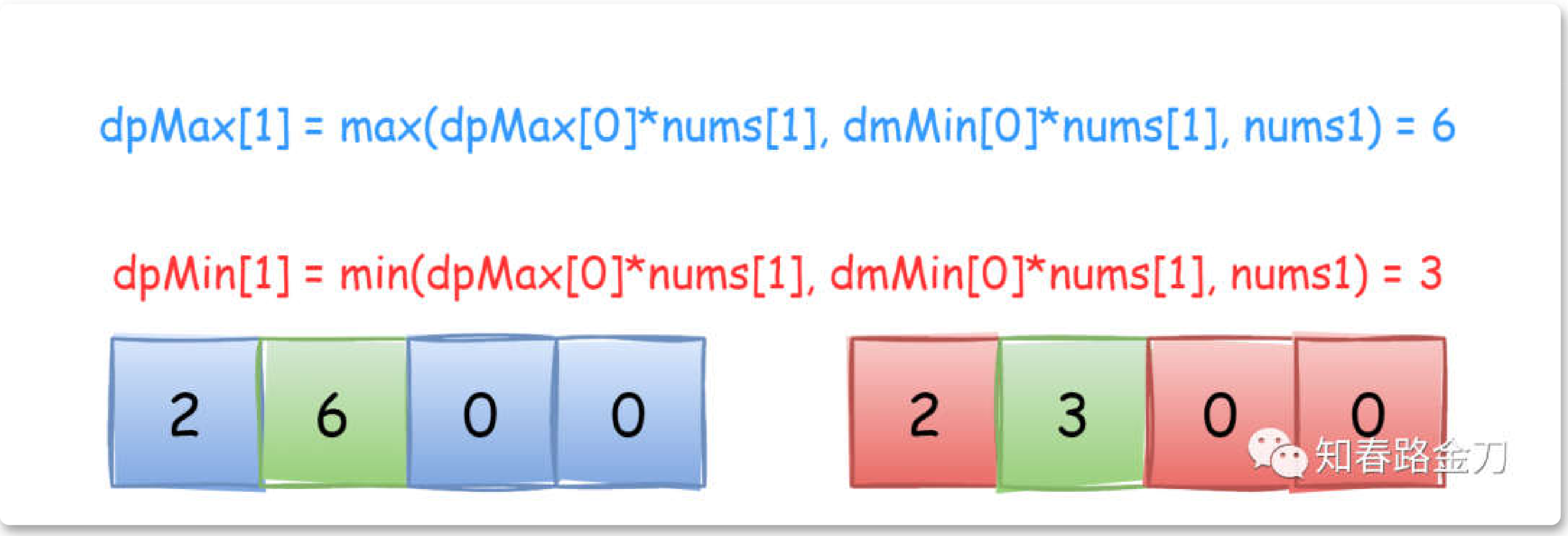
> dpMax[i] = max(nums[i], dpMax[i-1] * nums[i], *dpMin[i-1]* * nums[i]);
>
> dpMin[i] = min(nums[i], dpMax[i-1] * nums[i], *dpMin[i-1]* * nums[i]);

图解算法



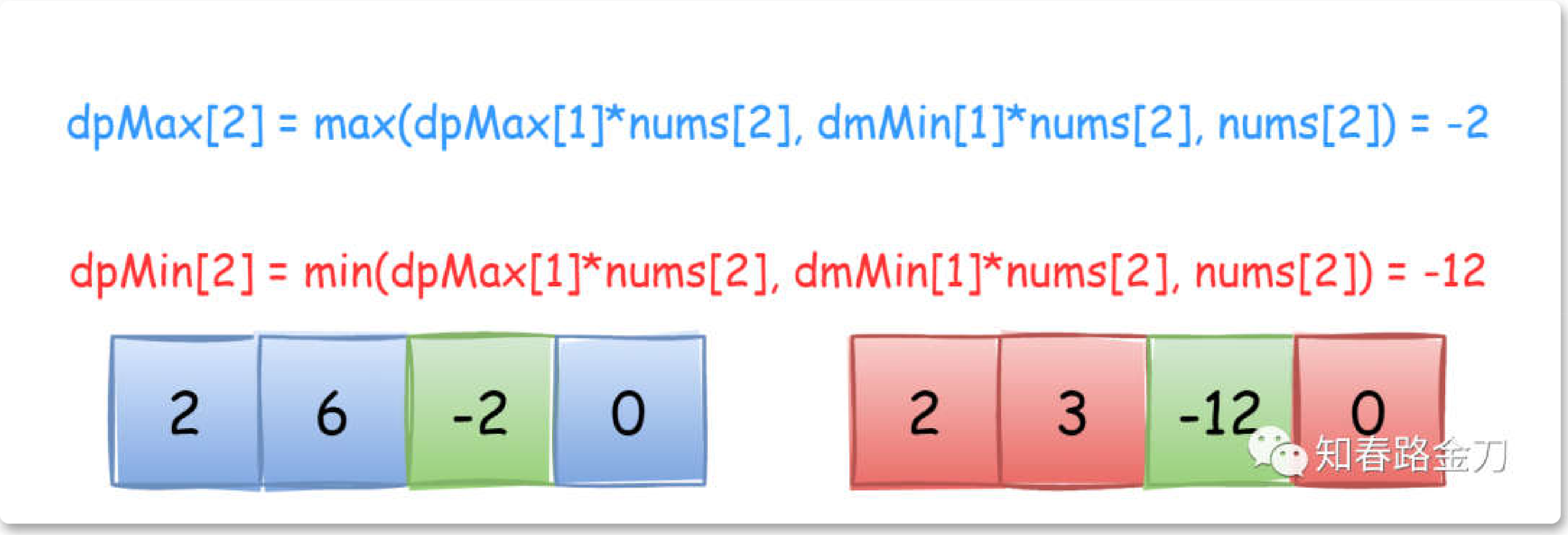
原始数组nums
2
3
-2
4
dpMax数组初始状态
2
0
0
0
dpMin数组初始状态
2
0
0
0
知春路金刀

「第一次迭代」



dpMax[1] = max(dpMax[0]*nums[1], dmMin[0]*nums[1], nums1) = 6
dpMin[1] = min(dpMax[0]*nums[1], dmMin[0]*nums[1], nums1) = 3
2
6
0
0
2
3
0
0
知春路金刀

「第二次迭代」



「第三次迭代」 这次迭代完后，达到了终止状态，最终的结果为 48

代码实现

```
class Solution {
    public int maxProduct(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        int[] dpMax = new int[nums.length];
        int[] dpMin = new int[nums.length];
        dpMax[0] = nums[0];
        dpMin[0] = nums[0];
        int result = nums[0];
        for (int i = 1; i < nums.length; i++) {
            dpMax[i] = Math.max(nums[i], Math.max(dpMax[i-1] * nums[i], dpMin[i-1] * nums[i]));
            dpMin[i] = Math.min(nums[i], Math.min(dpMax[i-1] * nums[i], dpMin[i-1] * nums[i]));
            result = Math.max(result, dpMax[i]);
        }
        return result;
    }
}
```

复杂度分析

- 时间复杂度：程序循环遍历了一次 nums，故渐进时间复杂度为 O(n)。
- 空间复杂度：new 了两个 dp 数组，故空间复杂度为 O(n)。

方法二：迭代(动态规划优化版)

思路和算法

在一维动态规划中我们需要 new 两个 dp 数组，导致了额外的空间占用。根据滚动数组思想，使用两个临时遍历就可以把 dp 数组优化掉。

代码实现

```java
class Solution {
    public int maxProduct(int[] nums) {
        int result = nums[0];
        int max = nums[0];
        int min = nums[0];
        for (int i = 1; i < nums.length; i++) {
            int maxt = Math.max(nums[i]*min, Math.max(nums[i]*max, nums[i]));
            int mint = Math.min(nums[i]*min, Math.min(nums[i]*max, nums[i]));
            result = Math.max(result, maxt);
            max = maxt;
            min = mint;
        }
        return result;
    }
}
```

复杂度分析

- 时间复杂度：循环遍历了一次 nums，故渐进时间复杂度为 O(n)
- 空间复杂度：优化后只使用常数个临时变量作为辅助空间，与 n 无关，故渐进空间复杂度为 O(1)

参考资料

[1] **原题链接:** https://leetcode-cn.com/problems/maximum-product-subarray/