0152. 乘积最大子数组

ITCharge 大约 3 分钟

- 标签：数组、动态规划
- 难度：中等

题目链接

- 0152. 乘积最大子数组 - 力扣

题目大意

描述：给定一个整数数组 `nums` 。

要求：找出数组中乘积最大的连续子数组（最少包含一个数字），并返回该子数组对应的乘积。

说明：

- 测试用例的答案是一个 32-位整数。
- **子数组**：数组的连续子序列。
- $1 \leq nums.length \leq 2 * 10^4$。
- $-10 \leq nums[i] \leq 10$。
- `nums` 的任何前缀或后缀的乘积都保证是一个 32-位整数。

示例：

- 示例 1：

```py
输入：nums = [2,3,-2,4]
输出：6
解释：子数组 [2,3] 有最大乘积 6。
```

- 示例 2：

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

解题思路

思路 1：动态规划

这道题跟「0053. 最大子序和 」有点相似，不过一个求的是和的最大值，这道题求解的是乘积的最大值。

乘积有个特殊情况，两个正数、两个负数相乘都会得到正数。所以求解的时候需要考虑负数的情况。

若想要最终的乘积最大，则应该使子数组中的正数元素尽可能的大，负数元素尽可能的小。所以我们可以维护一个最大值变量和最小值变量。

1. 划分阶段

按照子数组的结尾位置进行阶段划分。

2. 定义状态

定义状态 `dp_max[i]` 为：以第 i 个元素结尾的乘积最大子数组的乘积。

定义状态 `dp_min[i]` 为：以第 i 个元素结尾的乘积最小子数组的乘积。

3. 状态转移方程

- `dp_max[i] = max(dp_max[i - 1] * nums[i], nums[i], dp_min[i - 1] * nums[i])`
- `dp_min[i] = min(dp_min[i - 1] * nums[i], nums[i], dp_max[i - 1] * nums[i])`

4. 初始条件

- 以第 0 个元素结尾的乘积最大子数组的乘积为 `nums[0]` ，即 `dp_max[0] = nums[0]` 。
- 以第 0 个元素结尾的乘积最小子数组的乘积为 `nums[0]` ，即 `dp_min[0] = nums[0]` 。

5. 最终结果

根据状态定义，最终结果为 dp_{max} 中最大值，即乘积最大子数组的乘积。

思路 1：代码

```py
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        size = len(nums)
        dp_max = [0 for _ in range(size)]
        dp_min = [0 for _ in range(size)]
        dp_max[0] = nums[0]
        dp_min[0] = nums[0]
        ans = nums[0]
        for i in range(1, size):
            dp_max[i] = max(dp_max[i - 1] * nums[i], nums[i], dp_min[i - 1] *
nums[i])
            dp_min[i] = min(dp_min[i - 1] * nums[i], nums[i], dp_max[i - 1] *
nums[i])
        return max(dp_max)
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为整数数组 `nums` 的元素个数。
- **空间复杂度**：$O(n)$。

思路 2：动态规划 + 滚动优化

因为状态转移方程中只涉及到当前元素和前一个元素，所以我们也可以不使用数组，只使用两个变量来维护 $dp_{max}[i]$ 和 $dp_{min}[i]$。

思路 2：代码

```py
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        size = len(nums)
        max_num, min_num = nums[0], nums[0]
        ans = nums[0]
        for i in range(1, size):
            temp_max = max_num
            temp_min = min_num
            max_num = max(temp_max * nums[i], nums[i], temp_min * nums[i])
            min_num = min(temp_min * nums[i], nums[i], temp_max * nums[i])
            ans = max(max_num, ans)
        return ans
```

思路 2：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为整数 且 `nums` 的元素个数。
- **空间复杂度**：$O(1)$。