乘积最大子数组

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。

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int ans=nums[0];
        int maxf=nums[0];
        int minf=nums[0];
        for(int i=1;i<nums.size();i++)
        {
            int mx=maxf;//每一次循环记录上一次最大  最小
            int mn=minf;
            //更新最大  最小
            maxf=max(mx*nums[i],max(mn*nums[i],nums[i]));
            minf=min(mn*nums[i],min(mx*nums[i],nums[i]));
            ans=max(maxf,ans);
        }
        return ans;
    }
};
```

方法一：动态规划

思路和算法

如果我们用 $f_{\max}(i)$ 来表示以第 i 个元素结尾的乘积最大子数组的乘积，a 表示输入参数 `nums`，那么根据「53. 最大子序和」的经验，我们很容易推导出这样的状态转移方程：

$$f_{\max}(i) = \max_{i=1}^{n}\{f(i-1) \times a_i, a_i\}$$

它表示以第 i 个元素结尾的乘积最大子数组的乘积可以考虑 a_i 加入前面的 $f_{\max}(i-1)$ 对应的一段，或者单独成为一段，这里两种情况下取最大值。求出所有的 $f_{\max}(i)$ 之后选取最大的一个作为答案。

可是在这里，这样做是错误的。为什么呢？

因为这里的定义并不满足「最优子结构」。具体地讲，如果 $a = \{5, 6, -3, 4, -3\}$，那么此时 $f_{\max}$ 对应的序列是 $\{5, 30, -3, 4, -3\}$，按照前面的算法我们可以得到答案为 30，即前两个数的乘积，而实际上答案应该是全体数字的乘积。我们来想一想问题出在哪里呢？问题出在最后一个 -3 所对应的 $f_{\max}$ 的值既不是 -3，也不是 4×-3，而是 $5 \times 30 \times (-3) \times 4 \times (-3)$。所以我们得到了一个结论：当前位置的最优解未必是由前一个位置的最优解转移得到的。

我们可以根据正负性进行分类讨论。

考虑当前位置如果是一个负数的话，那么我们希望以它前一个位置结尾的某个段的积也是个负数，这样就可以负负得正，并且我们希望这个积尽可能「负得更多」，即尽可能小。如果当前位置是一个正数的话，我们更希望以它前一个位置结尾的某个段的积也是个正数，并且希望它尽可能地大。于是这里我们可以再维护一个 $f_{\min}(i)$，它表示以第 i 个元素结尾的乘积最小子数组的乘积，那么我们可以得到这样的动态规划转移方程：

$$f_{\max}(i) = \max_{i=1}^{n}\{f_{\max}(i-1) \times a_i, f_{\min}(i-1) \times a_i, a_i\}$$
$$f_{\min}(i) = \min_{i=1}^{n}\{f_{\max}(i-1) \times a_i, f_{\min}(i-1) \times a_i, a_i\}$$

它代表第 i 个元素结尾的乘积最大子数组的乘积 $f_{\max}(i)$，可以考虑把 a_i 加入第 $i-1$ 个元素结尾的乘积最大或最小的子数组的乘积中，二者加上 a_i，三者取大，就是第 i 个元素结尾的乘积最大子数组的乘积。第 i 个元素结尾的乘积最小子数组的乘积 $f_{\min}(i)$ 同理。

不难给出这样的实现：

C++ | Java

```
class Solution {
```