# 题目

一个数组的最小乘积定义为这个数组中最小值乘以数组的和。

比方说，数组[3,2,5]（最小值是2）的最小乘积为2 \* (3+2+5) = 2 \* 10 = 20。

给你一个正整数数组nums，请你返回nums任意非空子数组的最小乘积的 最大值。由于答案可能很大，请你返回答案对109 + 7取余的结果。

请注意，最小乘积的最大值考虑的是取余操作之前的结果。题目保证最小乘积的最大值在不取余的情况下可以用64位有符号整数保存。

子数组定义为一个数组的连续部分。

示例 1：

输入：nums = [1,2,3,2]

输出：14

解释：最小乘积的最大值由子数组 [2,3,2] （最小值是 2）得到。

2 \* (2+3+2) = 2 \* 7 = 14 。

示例 2：

输入：nums = [2,3,3,1,2]

输出：18

解释：最小乘积的最大值由子数组 [3,3] （最小值是 3）得到。

3 \* (3+3) = 3 \* 6 = 18 。

示例 3：

输入：nums = [3,1,5,6,4,2]

输出：60

解释：最小乘积的最大值由子数组 [5,6,4] （最小值是 4）得到。

4 \* (5+6+4) = 4 \* 15 = 60 。

提示：

1 <= nums.length <= 105

1 <= nums[i] <= 107

类似题目：

496. 下一个更大元素 I

503. 下一个更大元素 II

# 分析

## 方法一：单调栈

思路：

这道题可以使用单调栈来解决。具体步骤如下：

1、计算每个元素左边第一个小于当前元素的位置：从左到右遍历数组，对于每个元素，如果栈顶元素大于当前元素，说明当前元素是栈顶元素右边第一个小于栈顶元素的元素，记录下位置差，即为左边第一个小于当前元素的位置。否则，将当前元素入栈，继续遍历下一个元素。

2、计算每个元素右边第一个小于当前元素的位置：从右到左遍历数组，使用类似的方式计算右边第一个小于当前元素的位置。

3、计算以每个元素为最小值的子数组和：遍历数组，对于每个元素，计算以当前元素为最小值的子数组的和，即为当前元素值乘以左右位置差的乘积。

4、计算最小乘积的最大值：遍历所有计算出的子数组和，取最大值即为所求结果。

代码：

class Solution {

public:

int maxSumMinProduct(vector<int>& nums) {

const int MOD = 1e9 + 7;

int n = nums.size();

vector<long long> preSum(n + 1, 0); // 前缀和数组

for (int i = 0; i < n; ++i) {

preSum[i + 1] = preSum[i] + nums[i];

}

stack<int> st1; // 单调递增栈，存储元素下标

vector<int> left(n, -1); // 存储每个元素左边第一个小于当前元素的位置

for (int i = 0; i < n; ++i) {

while (!st1.empty() && nums[st1.top()] >= nums[i]) {

st1.pop();

}

if (!st1.empty()) {

left[i] = st1.top();

}

st1.push(i);

}

stack<int> st2; // 单调递增栈，存储元素下标

vector<int> right(n, n); // 存储每个元素右边第一个小于当前元素的位置

for (int i = n - 1; i >= 0; --i) {

while (!st2.empty() && nums[st2.top()] >= nums[i]) {

st2.pop();

}

if (!st2.empty()) {

right[i] = st2.top();

}

st2.push(i);

}

long long result = 0;

for (int i = 0; i < n; ++i) {

long long sum = preSum[right[i]] - preSum[left[i] + 1]; // 子数组和

result = max(result, sum \* nums[i]); // 更新最小乘积的最大值

}

return result % MOD;

}

};

复杂度分析：

时间复杂度：O(n)，其中n是数组nums的长度。计算数组left和right、前缀和以及答案都需要O(n)的时间。

空间复杂度：O(n)，即为单调栈和前缀和数组需要使用的空间。