# 题目

给定一个整数数组 arr，找到 min(b) 的总和，其中 b 的范围为 arr 的每个（连续）子数组。

由于答案可能很大，因此 返回答案模 10^9 + 7 。

示例 1：

输入：arr = [3,1,2,4]

输出：17

解释：

子数组为 [3]，[1]，[2]，[4]，[3,1]，[1,2]，[2,4]，[3,1,2]，[1,2,4]，[3,1,2,4]。

最小值为 3，1，2，4，1，1，2，1，1，1，和为 17。

示例 2：

输入：arr = [11,81,94,43,3]

输出：444

提示：

1 <= arr.length <= 3 \* 104

1 <= arr[i] <= 3 \* 104

# 分析

## 方法一：单调栈

思路：

这里使用了单调栈的思想来解决问题。单调栈是指栈中元素保持单调递增或单调递减的栈结构。

1、计算左边第一个小于当前元素的位置（每个元素都会记录一个与之对应的左侧小值）：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、计算右边第一个小于或等于当前元素的位置：从右到左遍历数组，使用类似的方式计算右边第一个小于或等于当前元素的位置。

3、计算子数组和：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。将所有子数组和相加即为最终结果。

这样可以保证每个元素都被考虑到，并且可以快速找到左右两边第一个小于或等于当前元素的位置，从而计算出以当前元素为最小值的子数组的和。

代码：

class Solution {

public:

int sumSubarrayMins(vector<int>& arr) {

const int MOD = 1e9 + 7;

int n = arr.size();

// 记录左侧、右侧较小值的数组

vector<long long> left(n), right(n);

// 单调栈，作为中间变量使用

stack<pair<int, int>> stLeft, stRight;

// 计算每个元素左边第一个小于它的元素的位置

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

while (!stLeft.empty() && arr[i] < stLeft.top().first) {

stLeft.pop();

}

// 当前i对应的值左侧较小的值记录在对应left数组的位置i

// 这样对数组中每个元素都建立了一个左侧小值的数组

left[i] = stLeft.empty() ? i + 1 : i - stLeft.top().second;

stLeft.push({arr[i], i});

}

// 计算每个元素右边第一个小于它的元素的位置

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

while (!stRight.empty() && arr[i] <= stRight.top().first) {

stRight.pop();

}

right[i] = stRight.empty() ? n - i : stRight.top().second - i;

stRight.push({arr[i], i});

}

// 计算结果

// 左侧较小值+右侧较小值+单个数值（即遍历数组每个元素）

long long ans = 0;

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

ans += (arr[i] \* left[i] \* right[i]);

ans %= MOD;

}

return ans;

}

};

复杂度分析：

时间复杂度：O(n)，其中n为数组的长度。利用单调栈求出每个元素为最小值的子序列长度需要的时间为O(n)，求出连续子数组的最小值的总和需要的时间为O(n)，因此总的时间复杂度为O(n)。

空间复杂度：O(n)。其中n为数组的长度。我们需要保存以每个元素为最小元素的子序列长度，所需的空间为O(n)。

## 方法二：动态规划