美丽塔I 题解

> 计算学部十大打卡活动——“龙舞编程新春会”编程打卡（2024-1-24）
>
> 力扣2865.美丽塔I

一、题目

给你一个长度为 n 下标从 **0** 开始的整数数组 maxHeights 。

你的任务是在坐标轴上建 n 座塔。第 i 座塔的下标为 i ， 高度为 heights[i] 。

如果以下条件满足，我们称这些塔是 **美丽** 的：

> 1. 1 <= heights[i] <= maxHeights[i]
> 2. heights 是一个 **山脉** 数组。

如果存在下标 i 满足以下条件，那么我们称数组 heights 是一个 **山脉** 数组：

> - 对于所有 0 < j <= i ， 都有 heights[j - 1] <= heights[j]
> - 对于所有 i <= k < n - 1 ， 都有 heights[k + 1] <= heights[k]

请你返回满足 **美丽塔** 要求的方案中，**高度和的最大值**。

示例 1：

> 输入：maxHeights = [5,3,4,1,1]
>
> 输出：13
>
> **解释**：和最大的美丽塔方案为 heights = [5,3,3,1,1] ，这是一个美丽塔方案，因为：
>
> - 1 <= heights[i] <= maxHeights[i]
>
> - heights 是个山脉数组，峰值在 i = 0 处。
>
> 13 是所有美丽塔方案中的最大高度和。

示例 2：

> 输入：maxHeights = [6,5,3,9,2,7]

> 输出：22
>
> **解释**：和最大的美丽塔方案为 heights = [3,3,3,9,2,2] ，这是一个美丽塔方案，因为：
>
> - 1 <= heights[i] <= maxHeights[i]
>
> - heights 是个山脉数组，峰值在 i = 3 处。
>
> 22 是所有美丽塔方案中的最大高度和。

示例 3：

> 输入：maxHeights = [3,2,5,5,2,3]
>
> 输出：18
>
> **解释**：和最大的美丽塔方案为 heights = [2,2,5,5,2,2] ，这是一个美丽塔方案，因为：
>
> - 1 <= heights[i] <= maxHeights[i]
>
> - heights 是个山脉数组，峰值在 i = 2 处。
>
> 18 是所有美丽塔方案中的最大高度和。

提示：

- 1 <= n == maxHeights <= 10^3
- 1 <= maxHeights[i] <= 10^9

二、题解

方法一 枚举

思路

由题意可知，**山状数组**直观理解为：从数组开头到结尾，元素先非递减再非递增或只递减或只递增。

而题目给出的输入——**maxHeights数组**，给出了山状数组每个元素的上限。

同时，对于山状数组中的最大值$height[i]$，可以发现，一定等于$maxHeights[i]$；否则，若小于$maxHeights[i]$，则可以使其增加到$maxHeights[i]$，得到更大的输出。

若确定山状数组中最大值的下标，则可以推出其他元素的最大值，得到最大的输出。

- 对于 $j \in [0, i-1]$ 时，此时$\max(heights[j]) = \min(heights[j+1], maxHeights[j])$;
- 对于 $j \in [i+1, n-1]$ 时，此时 $\max(heights[j]) = \min(heights[j-1], maxHeights[j])$;

故可以枚举以$maxHeights[i]$为**山顶**的**山状数组**之和求出最大的高度和。

代码

```
class Solution {
public:
    long long maximumSumOfHeights(vector<int>& maxHeights) {
        int n = maxHeights.size(); //数组大小
        int i, j, pre;
        long long temp, ans = 0; //由于数组每个值的取值最大为10^9，运算后可能会超过int表示范围，故用l
        for (i = 0; i < n; i++) {
            temp = maxHeights[i]; //temp变量保存以i为山顶下标的山状数组元素最大和
            pre = temp; //pre保存前一个或后一个元素最大值
            for (j = i + 1; j < n; j++) {
                pre = min(maxHeights[j], pre);
                temp += pre;
            }
            pre = maxHeights[i];
            for (j = i - 1; j >= 0; j--) {
                pre = min(maxHeights[j], pre);
                temp += pre;
            }
            ans = max(ans, temp);
        }
        return ans;
    }
};
```

复杂度分析

- 时间复杂度：$O(n^2)$，其中n表示给定数组的长度。枚举**山状**数组的**最大值**需要的时间为$O(n)$，给定最大值求数组元素的和需要的时间为$O(n)$，因此总的时间为$O(n^2)$。
- 空间复杂度：$O(1)$。

方法二 单调栈

思路

由方法一可知，若确定**山状**数组的**山顶**，则整个山状数组的所有元素的最大值即可确定，数组元素和的最大值也可以确定。

因此对于**山顶**下标i，可以将数组分为两部分处理，即保证数组的左侧构成非递减，右侧构成非递增。设区间$[0, i]$构成的非递减数组元素和最大值为$max1[i]$，区间$[i, n-1]$构成的非递增数组元素和最大值为$max2[i]$，此时构成的山状数组的元素之和即为$max1[i] + max2[i] - maxHeights[i]$。

可以使用**单调栈**来保证栈中数据的单调性，利用单调栈将连续子数组变为非递减或非递减。

对于左侧的**非递减**：将$maxHeights$ 顺序依次入栈，对于第i个元素来说，不断从栈顶弹出元素，直到栈顶元素小于等于$maxHeights[i]$。假设此时栈顶元素为$maxHeights[j]$，则区间$[j+1, i-1]$中的元素最多只能取到$maxHeights[i]$，则$max1[i] = max1[j] + (i - j) \times maxHeights[i]$;

同理，对于右侧的**非递增**：可以将$maxHeights$ 倒序依次入栈，转化为**非递减**，则$max2[i] = max2[j] + (j - i) \times maxHeights[i]$;

遍历每个**山顶**下标i，并计算出山状数组的元素之和最大值。

代码

```cpp
class Solution {
public:
    long long maximumSumOfHeights(vector<int>& maxHeights) {
        int n = maxHeights.size();
        int i;
        long long ans = 0;
        vector<long long> max1(n), max2(n);
        stack<int> st1, st2; //单调栈，存储maxHeights中元素下标
        for (i = 0; i < n; i++) {
            while (!st1.empty() && maxHeights[i] < maxHeights[st1.top()]) {
                st1.pop();
            }
            if (st1.empty()) {
                max1[i] = (long long)(i + 1) * maxHeights[i];//注意long long(i+1)
            } else {
                max1[i] = max1[st1.top()] +
                          (long long)(i - st1.top()) * maxHeights[i];
            }
            st1.emplace(i); //入栈（加速）
        }
        for (i = n - 1; i >= 0; i--) {
            while (!st2.empty() && maxHeights[i] < maxHeights[st2.top()]) {
                st2.pop();
            }
            if (st2.empty()) {
                max2[i] = (long long)(n - i) * maxHeights[i];
            } else {
                max2[i] = max2[st2.top()] +
                          (long long)(st2.top() - i) * maxHeights[i];
            }
            st2.emplace(i);
            ans = max(ans, max1[i] + max2[i] - maxHeights[i]); //将遍历每个i求出数组最大和的循环与:
        }
        return ans;
    }
};
```

复杂度分析

- 时间复杂度：$O(n)$，其中n表示给定数组的长度。利用单调栈求解该问题时，需要遍历两次数组，且在两个for循环中，每个元素最多入栈和出栈一次，所以需要总时间为$O(n)$。
- 空间复杂度：$O(n)$，其中n表示给定数组的长度。需要保存前缀和与后缀和，同时利用单调栈保存上一个更小的元素，需要的空间均为$O(n)$。