# 题目

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1：



输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]

输出：6

解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

示例 2：

输入：height = [4,2,0,3,2,5]

输出：9

提示：

n == height.length

0 <= n <= 3 \* 104

0 <= height[i] <= 105

# 分析

## 方法一：动态规划

class Solution {

public:

int trap(vector<int>& height) {

int n = height.size();

if (n == 0) {

return 0;

}

vector<int> leftMax(n);

leftMax[0] = height[0];

for (int i = 1; i < n; ++i) {

leftMax[i] = max(leftMax[i - 1], height[i]);

}

vector<int> rightMax(n);

rightMax[n - 1] = height[n - 1];

for (int i = n - 2; i >= 0; --i) {

rightMax[i] = max(rightMax[i + 1], height[i]);

}

int ans = 0;

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

ans += min(leftMax[i], rightMax[i]) - height[i];

}

return ans;

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是数组height的长度。计算数组leftMax和rightMax的元素值各需要遍历数组height一次，计算能接的雨水总量还需要遍历一次。

空间复杂度：O(n)，其中n是数组height的长度。需要创建两个长度为n的数组leftMax和rightMax。

## 方法二：单调栈

## 方法三：双指针

代码：

class Solution {

public:

int trap(vector<int>& height) {

int ans = 0;

int left = 0, right = height.size() - 1;

int leftMax = 0, rightMax = 0;

while (left < right) {

leftMax = max(leftMax, height[left]);

rightMax = max(rightMax, height[right]);

if (height[left] < height[right]) {

ans += leftMax - height[left];

++left;

} else {

ans += rightMax - height[right];

--right;

}

}

return ans;

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是数组height 的长度。两个指针的移动总次数不超过n。

空间复杂度：O(1)。只需要使用常数的额外空间。