# 题目

给定一个长度为 n 的整数数组 height 。有 n 条垂线，第 i 条线的两个端点是 (i, 0) 和 (i, height[i]) 。

找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

返回容器可以储存的最大水量。

说明：你不能倾斜容器。

示例 1：

图片包含 图标

描述已自动生成

输入：[1,8,6,2,5,4,8,3,7]

输出：49

解释：图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

示例 2：

输入：height = [1,1]

输出：1

提示：

n == height.length

2 <= n <= 105

0 <= height[i] <= 104

# 分析

## 方法一：双指针

要解决“盛最多水的容器”问题，核心是找到两条垂线，使其与x轴构成的容器能容纳最多的水。容器的水量由两条垂线的距离（宽度）和较短垂线的高度共同决定（水量 = 宽度 × 高度）。我们可以通过\*\*双指针法\*\*高效求解，避免暴力枚举的高时间复杂度。

解题思路

1、核心观察：容器的水量受限于“较短的垂线”和“两条垂线的距离”。若固定较短的垂线，无论如何移动较长的垂线，水量都不会增加（宽度减小，高度不变）。因此，我们应优先移动较短的垂线，尝试寻找更高的垂线以增加水量。

2、双指针策略：

- 初始化两个指针：left指向数组起始（最左侧垂线），right指向数组末尾（最右侧垂线）。

- 计算当前容器的水量（宽度 = right - left，高度 = min(height[left], height[right])），并更新最大水量。

- 移动较短的垂线：若height[left] < height[right]，则left右移（尝试寻找更高的左侧垂线）；否则right左移（尝试寻找更高的右侧垂线）。

- 重复上述步骤，直到left与right相遇（所有可能的组合已遍历）。

代码：

class Solution {

public:

int maxArea(vector<int>& height) {

int left = 0; // 左指针：初始指向最左侧

int right = height.size() - 1; // 右指针：初始指向最右侧

int max\_water = 0; // 存储最大水量

while (left < right) {

// 计算当前容器的宽度和高度

int width = right - left;

int current\_height = min(height[left], height[right]);

// 计算当前水量，并更新最大水量

int current\_water = width \* current\_height;

if (current\_water > max\_water) {

max\_water = current\_water;

}

// 移动较短的垂线（关键：优先寻找更高的垂线）

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

left++;

} else {

right--;

}

}

return max\_water;

}

};

代码解释

1、初始化：left从数组开头出发，right从数组结尾出发，覆盖最宽的初始容器（最大可能宽度）。

2、水量计算：每次循环计算当前容器的水量，通过min(height[left], height[right])取较短垂线的高度，确保水量计算符合物理逻辑（水不会溢出）。

3、指针移动：始终移动较短的垂线，是因为移动较长的垂线只会导致“宽度减小、高度不变”，水量必然减少；而移动较短的垂线可能找到更高的垂线，从而增加水量。

4、终止条件：当left与right相遇时，所有可能的垂线组合已遍历，此时max\_water即为最大水量。

复杂度分析

- 时间复杂度：O(n)，仅遍历数组一次（双指针共移动 n-1 次），远优于暴力枚举的 O(n²)。

- 空间复杂度：O(1)，仅使用常数额外空间（left、right、max\_water等变量），符合空间效率要求。