11. 盛水最多的容器

地址：盛水最多的容器

题目：

- English

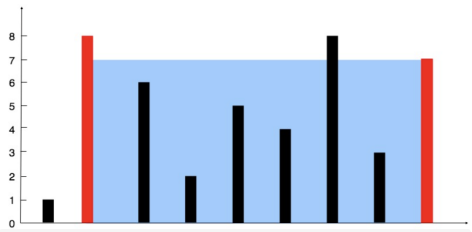
11. Container With Most Water

难度 中等 2588

Given n non-negative integers $a_1, a_2, ..., a_n$, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of the line i is at (i, a_i) and $(i, 0)$. Find two lines, which, together with the x-axis forms a container, such that the container contains the most water.

Notice that you may not slant the container.

Example 1:



```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.
```

Example 2:

```
Input: height = [1,1]
Output: 1
```

Example 3:

```
Input: height = [4,3,2,1,4]
Output: 16
```

Example 4:

```
Input: height = [1,2,1]
Output: 2
```

Constraints:

- n == height.length
- $2 <= n <= 10^5$
- $0 <= height[i] <= 10^4$

- 中文：

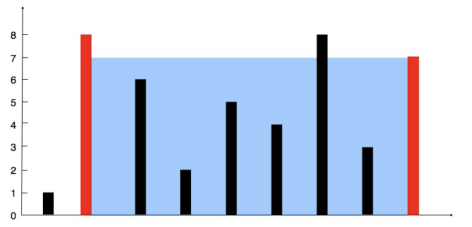
11. 盛最多水的容器

难度 中等 2588

给你 n 个非负整数 $a_1, a_2, ..., a_n$，每个数代表坐标中的一个点 (i, a_i)。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, a_i) 和 $(i, 0)$。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。

说明：你不能倾斜容器。

示例 1：



```
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
```

示例 3：

```
输入：height = [4,3,2,1,4]
输出：16
```

示例 4：

```
输入：height = [1,2,1]
输出：2
```

提示：

- n = height.length
- $2 <= n <= 3 * 10^4$
- $0 <= height[i] <= 3 * 10^4$

思路 1：暴力枚举，超出时间限制

分析

★ 从 left bar i 逐个迭代尝试到 right bar j

- area = width * high
- width = j - i
- high = min(i - height, j - height) - 最多能装多少水由最小的高度决定

Code

```java
// Java
// Did Time : 2021 - 07 - 07


class Solution {


  public int maxArea(int[] height) {
    int max = 0;


    // 重要：二次逐个遍历数组，要注意左右边界，不要有反复值，需要牢记
    for (int i = 0; i < height.length - 1; i++) {
      for (int j = i + 1; j < height.length; j++) {
        int area = (j - i) * Math.min(height[i], height[j]);
          max = Math.max(max, area);
      }
    }
    return max;
  }
}
```

复杂度分析 ：

- 时间复杂度：O(n ^ 2)
- 空间复杂度：O（1）

* 思路 2 ： 双指针，左右夹逼 （重要）

分析

- 双指针，左右边界，左右夹逼，向中间收敛。
- 左右边界选在最两边，向中间收敛，找到相对高度比较高的两边界，计算面积，直到两边界相遇，返回面积最大者。
- 说明：左右边界选在最两边，宽度最大，但高度不一定最高；向中间收敛，只关注最高的边界，将其作为新的边界。因为，宽度在缩小，高度若还缩小，面积则一定缩小，就不用考虑了。

代码

```java
// Java
// Did Time : 2021 - 07 - 07

```

```java

class Solution {


  public int maxArea(int[] height) {
    int maxArea = 0;
    for (int i = 0, j = height.length - 1; i < j;) {
      // i 左边界，向右走；j 右边界，向左走；谁小谁先走，作为迭代条件
      // height[i++] : 当 i 的高度小于 j 的高度时， i++ 既可以使 i 向左走，又可以得到在计算面积时 i 的实际坐标值
      // height[j--] : 当 i 的高度大于 j 的高度时， j-- 使 j 向右走，但在计算面积时 j 的实际坐标值多减了 1
      // j + 1 : 计算面积时，为了迭代 j 的实际值多减少了1
      int minHeight = height[i] < height[j] ? height[i++] : height[j--];


      // area = ( j + 1 - i ) * minHeight;
      maxArea = Math.max(maxArea, (j + 1 - i) * minHeight);
    }
    return maxArea;
  }
}
```

复杂度分析

- 时间复杂度：O（n）
- 空间复杂度：O（1）