Description | Accepted | Editorial | Solutions | Submissions

11. Container With Most Water

Solved

Medium | Topics | Companies | Hint

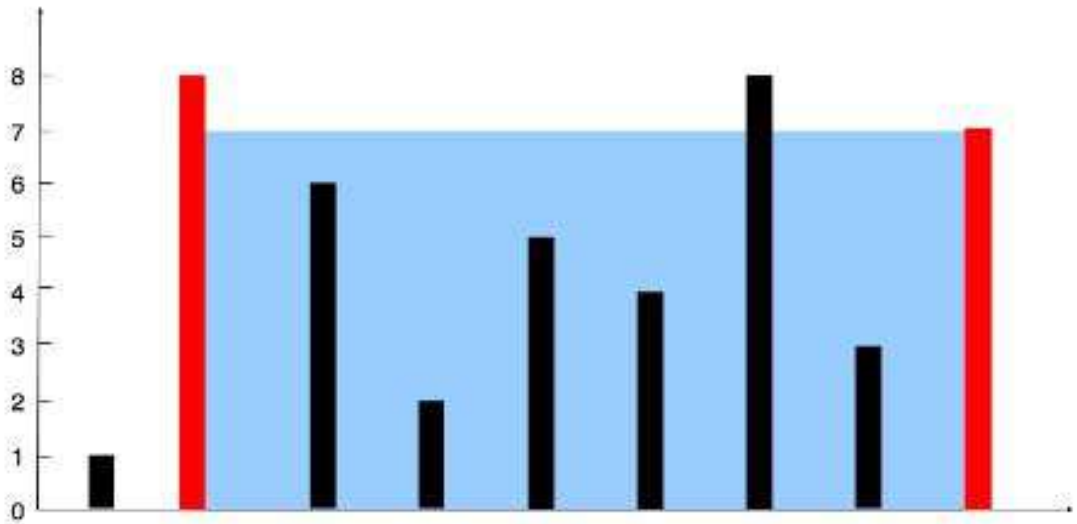
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i^th` line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store.*

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.
```

Container With Most Water >

Input height = [1, 8, 6, 2, 5, 4, 8, 3, 7]
Output 49

Two pointer Approach:

Start From left (0) and right (length-1).
Calculate the Stored Water ((min(height[left], height[right])) * (right - left)) width
If it is greater than max which was initialy at 0
update the max with storedwater.
IF height[left] > height[right] move right
pointer (right--)
else move left pointer (left++).
Continue this process in a loop until
left < right.
After that return max.

ava Auto

```java
    public int maxArea(int[] height) {
        int left = 0;
        int right = height.length - 1;
        int max = 0;
        while (left < right) {

            int temp;

            int width = right - left;

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

                temp = width * height[right];
                if (temp > max)
                    max = temp;
                right--;
            } else {
                temp = width * height[left];

                if (temp > max)
                    max = temp;
                left++;
            }

        }
        return max;
    }
}
```

aved Ln 2, Col 39

Description | Accepted × | Editorial | Solutions | Submissions

All Submissions

Accepted

Azan imtiaz submitted at Aug 01, 2024 16:36

Editorial | Solution

Runtime

3 ms | Beats 95.17%

Analyze Complexity

Memory

58.59 MB | Beats 7.11%

Code | Java

```
class Solution {
    public int maxArea(int[] height) {

        int left=0;
```

125. Valid Palindrome

Solved

Easy | Topics | Companies

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string `s`, return `true` *if it is a* ***palindrome****, or* `false` *otherwise.*

Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

Example 2:

```
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
```

Example 3:

```
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.
```

Constraints:

- $1 \le s.length \le 2 \times 10^5$
- `s` consists only of printable ASCII characters.

Valid Palindrome

Approach>

Simply used two pointers approach from left and right.

Inside outer loop apply one loop to

avoid non alphanumeric character on left side.
And one loop to avoid nonalphanumeric on right side.
After that convert both left and right character to lowercase and the compare both characters.

Description | Accepted × | Editorial | Solutions | Submissions

← All Submissions

Accepted

Azan imtiaz submitted at Aug 01, 2024 17:11

Editorial | Solution

Runtime

2 ms | Beats 99.07%

Analyze Complexity

Memory

42.89 MB | Beats 72.98%

Code | Java

```
class Solution {
    public boolean isPalindrome(String s) {
        int l=0;
        int r=s.length()-1;
         while(l<r){
            while (l < r && !Character.isLetterOrDigit(s.charAt(l))) {
                l++;
            }
```

View more

Code

Java Auto

```java
class Solution {
    public boolean isPalindrome(String s) {
        int l=0;
        int r=s.length()-1;
         while(l<r){
            while (l < r && !Character.isLetterOrDigit(s.charAt(l))) {
                l++;
            }

            // Move right pointer to the previous alphanumeric character
            while (l < r && !Character.isLetterOrDigit(s.charAt(r))) {
                r--;
            }

            // Compare characters
            if (Character.toLowerCase(s.charAt(l)) != Character.toLowerCase(s.charAt(r))) {
                return false;
            }
            l++;
            r--;
         }
         return true;
    }
}
```

Saved

Ln 1, Col 1

Testcase | Test Result

Case 1 Case 2 Case 3