

1 LI TwoSum.JAVA

The screenshot shows the LeetCode interface for the "Two Sum" problem. The problem description on the left explains that given an array of integers `nums` and an integer `target`, the goal is to return indices of two numbers that add up to the target. It includes three examples with their inputs, outputs, and explanations. Constraints specify the array length and the range of values. A follow-up question asks for a more efficient algorithm.

The code editor on the right contains a Java solution. It defines a `TwoSum` class with a `find` method that uses a `HashMap` to store the complement of each element. The `main` method uses a `Scanner` to read input and prints the result.

```
import java.util.*;

class Solution {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[] {map.get(complement), i};
            }
            map.put(nums[i], i);
        }
        throw new IllegalArgumentException("No solution found");
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] nums = new int[n];
    }
}
```

The test results section shows that the solution is "Accepted" with a runtime of 0 ms. It displays the input `nums = [2, 7, 11, 15]` and `target = 9`, and the output `[0, 1]`.

2 LI AddTwoNumbers.JAVA

The screenshot shows the LeetCode interface for the "Add Two Numbers" problem. The problem description on the left explains that two non-empty linked lists representing non-negative integers are given, with digits stored in reverse order. The goal is to add the two numbers and return the sum as a linked list. It includes three examples with their inputs, outputs, and explanations. Constraints specify the range of nodes and the absence of leading zeros.

The code editor on the right contains a Java solution. It defines a `ListNode` class and a `addTwoNumbers` method that iterates through both linked lists, adding corresponding digits and handling the carry. The `main` method uses a `Scanner` to read input and prints the result.

```
class ListNode {
    int val;
    ListNode next;
    ListNode() {}
    ListNode(int val) { this.val = val; }
    ListNode(int val, ListNode next) { this.val = val; this.next = next; }
}

class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode curr = dummy;
        int carry = 0;
        while (l1 != null || l2 != null || carry != 0) {
            int sum = 0;
            if (l1 != null) {
                sum += l1.val;
                l1 = l1.next;
            }
            if (l2 != null) {
                sum += l2.val;
                l2 = l2.next;
            }
            carry = sum / 10;
            curr.next = new ListNode(sum % 10);
            curr = curr.next;
        }
        return dummy.next;
    }
}
```

The test results section shows that the solution is "Accepted" with a runtime of 0 ms. It displays the input `l1 = [2, 4, 3]` and `l2 = [5, 6, 4]`, and the output `[7, 0, 8]`.

3 LI

LongestSubstringWithoutRepeatingCharacters.JAVA

5. Longest Palindromic Substring

Given a string *s*, return the longest palindromic substring in *s*.

Example 1:
Input: *s* = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.

Example 2:
Input: *s* = "cbbd"
Output: "bb"

Constraints:

- s* has a length of up to 1000.
- s* consist of only digits and English letters.

Accepted: 4,113,358/11,341 Acceptance Rate: 36.3%

```
Java
class Solution {
    public String longestPalindrome(String s) {
        if (s == null || s.length() == 0) return "";
        int start = 0, end = 0;
        for (int i = 0; i < s.length(); i++) {
            int left = expandFromCenter(i, i);
            int right = expandFromCenter(i, i + 1);
            int len = Math.max(left, right);
            if (len > end - start) {
                start = i - (len - 1) / 2;
                end = i + (len - 1) / 2;
            }
        }
        return s.substring(start, end + 1);
    }
    private int expandFromCenter(int left, int right) {
        while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
            left--;
            right++;
        }
        return right - left - 1;
    }
}
```

6. ZigzagConversion.JAVA

6. Zigzag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this (you may want to display this pattern in a fixed font for better legibility):

```
P A Y P A L I S H I R I N G
A P L S I I R
Y I R
```

And then read line by line: "PAYPALISHIRING"

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string s, int numRows);
```

Example 1:
Input: *s* = "PAYPALISHIRING", *numRows* = 3
Output: "PAYPALISHIRING"

Example 2:
Input: *s* = "PAYPALISHIRING", *numRows* = 4
Output: "PINALISYAIPRI"

Example 3:
Input: *s* = "A", *numRows* = 1
Output: "A"

Constraints:

- s* has a length of up to 1000.
- s* consists of English letters (lower-case and upper-case), `" "` and `","`.
- 1 <= numRows <= 1000*.

Accepted: 1,885,913/11,014 Acceptance Rate: 52.2%

```
Java
class Solution {
    public String convert(String s, int numRows) {
        if (numRows == 1 || s.length() <= numRows) return s;
        StringBuilder[] rows = new StringBuilder[numRows];
        for (int i = 0; i < numRows; i++) rows[i] = new StringBuilder();
        int curRow = 0;
        boolean goingDown = false;
        for (char c : s.toCharArray()) {
            rows[curRow].append(c);
            if (curRow == 0 || curRow == numRows - 1) goingDown = !goingDown;
            curRow = goingDown ? 1 + curRow : numRows - 1 - curRow;
        }
        StringBuilder result = new StringBuilder();
        for (StringBuilder row : rows) result.append(row);
        return result.toString();
    }
}
```

7. Reverse Integer.JAVA

7. Reverse Integer

Medium

Topics

Companies

Given a signed 32-bit integer x , return y with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range $[-2^{31}, 2^{31} - 1]$, then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1:
Input: $x = 123$
Output: 321

Example 2:
Input: $x = -123$
Output: -321

Example 3:
Input: $x = 120$
Output: 21

Constraints:
 $-2^{31} \leq x \leq 2^{31} - 1$

Solve this question in a real interview before? 1/5

Accepted 4,343,250/14,384 Acceptance Rate 30.7%

Topics

Companies

Similar Questions

Discussion (587)

Copyright © 2025 LeetCode. All rights reserved.

Code

```

1 class Solution {
2     public int reverse(int x) {
3         int rev = 0;
4         while (x != 0) {
5             int pop = x % 10;
6             x /= 10;
7             // overflow check
8             if (rev > Integer.MAX_VALUE/10 || (rev == Integer.MAX_VALUE/10 && pop > 7)) return 0;
9             if (rev < Integer.MIN_VALUE/10 || (rev == Integer.MIN_VALUE/10 && pop < -8)) return 0;
10            rev = rev * 10 + pop;
11        }
12        return rev;
13    }
14 }

```

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

$x =$

123

Output

321

Expected

321

Contribute a Testcase

8. String to Integer (atoi).JAVA

8. String to Integer (atoi)

Medium

Topics

Companies

Implement the `myAtoi(String s)` function, which converts a string to a 32-bit signed integer.

The algorithm for `myAtoi(String s)` is as follows:

- Whitespace:** Ignore any leading whitespace (`' '`).
- Sign:** Determine the sign by checking if the next character is `'+'` or `'-'`, assuming positivity if neither present.
- Conversion:** Read the integer by skipping leading zeros until a non-digit character is encountered or the end of the string is reached. If no digits were read, then the result is 0.
- Rounding:** If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then round the integer to remain in the range. Specifically, integers less than -2^{31} should be rounded to -2^{31} , and integers greater than $2^{31} - 1$ should be rounded to $2^{31} - 1$.

Return the integer as the final result.

Example 1:
Input: $s = "42"$
Output: 42
Explanation:
The underlined characters are what is read in and the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
Step 2: "42" (no characters read because there is neither a '+' nor '-' sign)
Step 3: "42" ("42" is read in)

Example 2:
Input: $s = " -42"$
Output: -42
Explanation:
Step 1: " -42" (leading whitespace is read and ignored)
Step 2: "-42" ('-' is read, so the result should be negative)
Step 3: "-42" ("42" is read in, leading zeros ignored in the result)

Example 3:
Input: $s = "13371 067"$
Output: 1337
Explanation:
Step 1: "13371 067" (leading zeros are read and ignored)

Code

```

1 // Convert string to integer
2 // 1. Skip leading whitespaces
3 // 2. Check for sign
4 // 3. Convert digits to integer
5 // 4. Check for overflow
6 // 5. Return the result
7
8 public int myAtoi(String s) {
9     // Skip leading whitespaces
10    int i = 0;
11    while (i < s.length() && s.charAt(i) == ' ') {
12        i++;
13    }
14
15    // Check for sign
16    int sign = 1;
17    if (i < s.length() && (s.charAt(i) == '+' || s.charAt(i) == '-')) {
18        sign = (s.charAt(i) == '-') ? -1 : 1;
19        i++;
20    }
21
22    // Convert digits to integer
23    long result = 0;
24    while (i < s.length() && Character.isDigit(s.charAt(i))) {
25        int digit = s.charAt(i) - '0';
26
27        // Check for overflow
28        if (result > (Integer.MAX_VALUE - digit) / 10) {
29            return (sign == 1) ? Integer.MAX_VALUE : Integer.MIN_VALUE;
30        }
31
32        result = result * 10 + digit;
33    }
34
35    return result * sign;
36 }

```

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3 Case 4 Case 5

Input

$s =$

"42"

Output

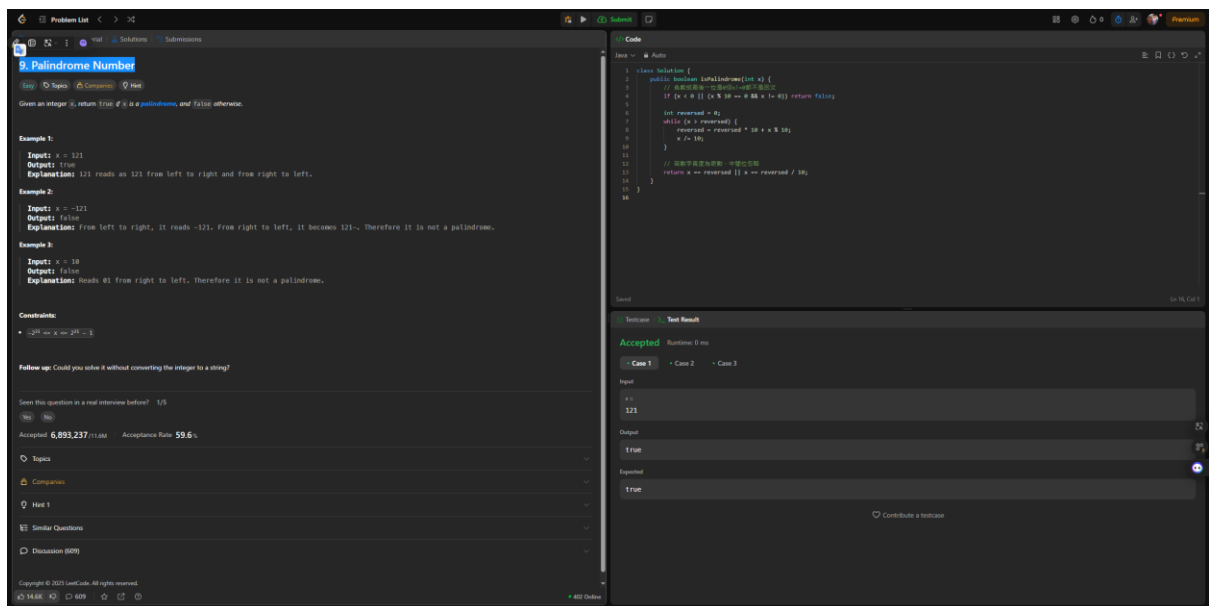
42

Expected

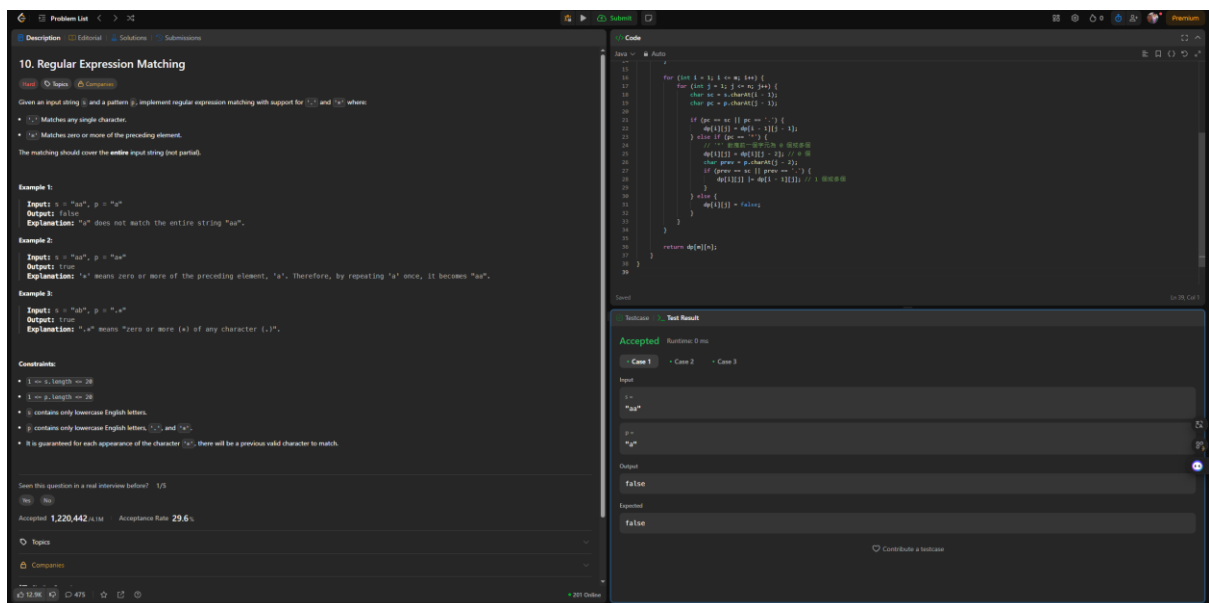
42

Contribute a Testcase

9. Palindrome Number.JAVA



10. Regular Expression Matching.JAVA



11. Container With Most Water.JAVA

Problem List

Description

Editorial

Solutions

Submissions

11. Container With Most Water

Medium

Types

Companies

Notes

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.

Example 2:

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

Constraints:

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

Seen this question in a real interview before? 1/5

Accepted **4,413,661** / 4448 Acceptance Rate: 58.2%

<https://leetcode.com/problems/container-with-most-water/>

Run

Ctrl

Submit

Code

Auto

```

1 class Solution {
2     public int maxArea(int[] height) {
3         int left = 0, right = height.length - 1;
4         int maxArea = 0;
5
6         while (left < right) {
7             int h = Math.min(height[left], height[right]);
8             int w = right - left;
9             maxArea = Math.max(maxArea, h * w);
10
11             // 移动短板/移动长板
12             if (height[left] < height[right]) {
13                 left++;
14             } else {
15                 right--;
16             }
17         }
18
19         return maxArea;
20     }
21 }

```

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

`height = [1,8,6,2,5,4,8,3,7]`

Output

`49`

Expected

`49`

Contribute a Testcase

保存到我的草稿 在浏览器中打开此题单

12. Integer to Roman.JAVA

Problem List

Description

Editorial

Solutions

Submissions

12. Integer to Roman

Medium

Types

Companies

Seven different symbols represent Roman numerals with the following values:

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Roman numerals are formed by appending the conversions of decimal place values from highest to lowest. Converting a decimal place value into a Roman numeral has the following rules:

- If the value does not start with 4 or 9, the symbol of the maximal value that can be subtracted from the input, append that symbol to the result, subtract its value, and convert the remainder to a Roman numeral.
- If the value starts with 4 or 9 use the **subtractive form** representing one symbol subtracted from the following symbol, for example, 4 is 1 (I) less than 5 (V): `IV` and 9 is 1 (I) less than 10 (X): `IX`. Only the following subtractive forms are used: 4 (`IV`), 9 (`IX`), 40 (`XL`), 90 (`XC`), 400 (`CD`) and 900 (`CM`).
- Only powers of 10 (`I`, `V`, `X`, `L`, `C`, `D`, `M`) can be appended consecutively at most 3 times to represent multiples of 10. You cannot append 5 (`V`), 50 (`L`), or 500 (`D`) multiple times. If you need to append a symbol 4 times use the **subtractive form**.

Given an integer, convert it to a Roman numeral.

Example 1:

Input: `num = 3749`
Output: `"MMDCCLXXIV"`

Explanation:

3000 = MMM as 1000 (M) + 1000 (M) + 1000 (M)
700 = DCC as 500 (D) + 100 (C) + 100 (C)
40 = XL as 10 (X) less of 50 (L)
9 = IX as 1 (I) less of 10 (X)
Note: 49 is not 1 (I) less of 50 (L) because the conversion is based on decimal places

Example 2:

Input: `num = 58`
Output: `"LVIII"`

Explanation:

Run

Ctrl

Submit

Code

Auto

```

1 class Solution {
2     public String intToRoman(int num) {
3         // Roman numerals: I, V, X, L, C, D, M
4         int[] values = {1000, 900, 500, 400, 100, 50, 10, 5, 1};
5         String[] symbols = {"M", "CM", "D", "CD", "C", "L", "XL", "X", "LX", "V", "IV", "I"};
6
7         StringBuilder sb = new StringBuilder();
8
9         for (int i = 0; i < values.length; num >= values[i]; i++) {
10             while (num >= values[i]) {
11                 num -= values[i];
12                 sb.append(symbols[i]);
13             }
14         }
15
16         return sb.toString();
17     }
18 }

```

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

`num = 3749`

Output

`"MMDCCLXXIV"`

Expected

`"MMDCCLXXIV"`

Contribute a Testcase

保存到我的草稿 在浏览器中打开此题单

13. Roman to Integer.JAVA

