LeetCode 75

Array / String		
Merge Strings Alternately	Solution	Easy
Greatest Common Divisor of Strings	Solution	Easy
Kids With the Greatest Number of Candies	Solution	Easy
Can Place Flowers	Solution	Easy
Reverse Vowels of a String	Solution	Easy
Reverse Words in a String	Solution	Medium
Product of Array Except Self	Solution	Medium
Increasing Triplet Subsequence	Solution	Medium
String Compression	Solution	Medium

Two Pointers		
Move Zeroes	Solution	Easy
Is Subsequence	Solution	Easy
Container With Most Water	Solution	Medium
Max Number of K-Sum Pairs	Solution	Medium

Sliding Window		
Maximum Average Subarray I	Solution	Easy
Maximum Number of Vowels in a Substring of Given Length	Solution	Medium
Max Consecutive Ones III	Solution	Medium
Longest Subarray of 1's After Deleting One Element	Solution	Medium

Prefix Sum		
Find the Highest Altitude	Solution	Easy
Find Pivot Index	Solution	Easy

Hash Map / Set		
Find the Difference of Two Arrays	Solution	Easy
Unique Number of Occurrences	Solution	Easy
Determine if Two Strings Are Close	Solution	Medium
Equal Row and Column Pairs	Solution	Medium

Stack		
Removing Stars From a String	Solution	Medium
Asteroid Collision	Solution	Medium
Decode String	Solution	Medium

Queue		
Number of Recent Calls	Solution	Easy
Dota2 Senate	Solution	Medium

Linked List		
Delete the Middle Node of a Linked List	Solution	Medium
Odd Even Linked List	Solution	Medium
Reverse Linked List	Solution	Easy
Maximum Twin Sum of a Linked List	Solution	Medium

Binary Tree - DFS		
Maximum Depth of Binary Tree	Solution	Easy
Leaf-Similar Trees	Solution	Easy
Count Good Nodes in Binary Tree	Solution	Medium
Path Sum III	Solution	Medium
Longest ZigZag Path in a Binary Tree	Solution	Medium
Lowest Common Ancestor of a Binary Tree	Solution	Medium

Binary Tree - BFS		
Binary Tree Right Side View	Solution	Medium
Maximum Level Sum of a Binary Tree	Solution	Medium

Binary Search Tree		
Search in a Binary Search Tree	Solution	Easy
Delete Node in a BST	Solution	Medium

Graphs - DFS		
Keys and Rooms	Solution	Medium
Number of Provinces	Solution	Medium
Reorder Routes to Make All Paths Lead to the City Zero	Solution	Medium
Evaluate Division	Solution	Medium

Graphs - BFS		
Nearest Exit from Entrance in Maze	Solution	Medium
Rotting Oranges	Solution	Medium

Heap / Priority Queue		
Kth Largest Element in an Array	Solution	Medium
Smallest Number in Infinite Set	Solution	Medium
Maximum Subsequence Score	Solution	Medium
Total Cost to Hire K Workers	Solution	Medium

Binary Search		
Guess Number Higher or Lower	Solution	Easy
Successful Pairs of Spells and Potions	Solution	Medium
Find Peak Element	Solution	Medium
Koko Eating Bananas	Solution	Medium

Backtracking		
Letter Combinations of a Phone Number	Solution	Medium
Combination Sum III	Solution	Medium

DP - 1D		
N-th Tribonacci Number	Solution	Easy
Min Cost Climbing Stairs	Solution	Easy
House Robber	Solution	Medium
Domino and Tromino Tiling	Solution	Medium

DP - Multidimensional		
Unique Paths	Solution	Medium
Longest Common Subsequence	Solution	Medium
Best Time to Buy and Sell Stock with Transaction Fee	Solution	Medium
Edit Distance	Solution	Medium

Bit Manipulation		
Counting Bits	Solution	Easy
Single Number	Solution	Easy
Minimum Flips to Make a OR b Equal to c	Solution	Medium

Trie		
Implement Trie (Prefix Tree)	Solution	Medium
Search Suggestions System	Solution	Medium

Intervals		
Non-overlapping Intervals	Solution	Medium
Minimum Number of Arrows to Burst Balloons	Solution	Medium

Monotonic Stack

Daily Temperatures	Solution	Medium
Online Stock Span	Solution	Medium

Array/String

Description | Editorial | Solutions | Submissions

1768. Merge Strings Alternately

Solved

Easy | Topics | Companies | Hint

You are given two strings `word1` and `word2`. Merge the strings by adding letters in alternating order, starting with `word1`. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return *the merged string.*

Example 1:

```
Input: word1 = "abc", word2 = "pqr"
Output: "apbqcr"
Explanation: The merged string will be merged as so:
word1:  a   b   c
word2:    p   q   r
merged: a p b q c r
```

Example 2:

```
Input: word1 = "ab", word2 = "pqrs"
Output: "apbqrs"
Explanation: Notice that as word2 is longer, "rs" is appended to the end.
word1:  a   b
word2:    p   q   r   s
merged: a p b q   r   s
```

Example 3:

```
Input: word1 = "abcd", word2 = "pq"
Output: "apbqcd"
Explanation: Notice that as word1 is longer, "cd" is appended to the end.
word1:  a   b   c   d
word2:    p   q
merged: a p b q c   d
```

Code

Java | Auto

```java
class Solution {
    public String mergeAlternately(String word1, String word2) {
        StringBuilder sb = new StringBuilder();
        int i = 0, j = 0;

        while (i < word1.length() && j < word2.length()) {
            sb.append(word1.charAt(i++));
            sb.append(word2.charAt(j++));
        }

        if (i < word1.length()) {
            sb.append(word1.substring(i));
        }

        if (j < word2.length()) {
            sb.append(word2.substring(j));
        }

        return sb.toString();
    }
}
```

Description | Editorial | Solutions | Submissions

1071. Greatest Common Divisor of Strings

Solved

Easy | Topics | Companies | Hint

For two strings `s` and `t`, we say "`t` divides `s`" if and only if `s = t + t + t + ... + t + t` (i.e., `t` is concatenated with itself one or more times).

Given two strings `str1` and `str2`, return *the largest string* `x` *such that* `x` *divides both* `str1` *and* `str2`.

Example 1:

```
Input: str1 = "ABCABC", str2 = "ABC"
Output: "ABC"
```

Example 2:

```
Input: str1 = "ABABAB", str2 = "ABAB"
Output: "AB"
```

Example 3:

```
Input: str1 = "LEET", str2 = "CODE"
Output: ""
```

Constraints:

- `1 <= str1.length, str2.length <= 1000`
- `str1` and `str2` consist of English uppercase letters.

Code

Java | Auto

```java
class Solution {
    public String gcdOfStrings(String str1, String str2) {
        if (!(str1 + str2).equals(str2 + str1)) {
            return "";
        }

        int gcdLength = gcd(str1.length(), str2.length());

        return str1.substring(0, gcdLength);
    }

    private int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }

}
```

Description | Editorial | Solutions | Submissions

1431. Kids With the Greatest Number of Candies

Solved

Easy | Topics | Companies | Hint

There are `n` kids with candies. You are given an integer array `candies`, where each `candies[i]` represents the number of candies the `i`th kid has, and an integer `extraCandies`, denoting the number of extra candies that you have.

Return *a boolean array* `result` *of length* `n`, *where* `result[i]` *is* `true` *if, after giving the* `i`th *kid all the* `extraCandies`, *they will have the* ***greatest*** *number of candies among all the kids, or* `false` *otherwise*.

Note that **multiple** kids can have the **greatest** number of candies.

Example 1:

```
Input: candies = [2,3,5,1,3], extraCandies = 3
Output: [true,true,true,false,true]
Explanation: If you give all extraCandies to:
- Kid 1, they will have 2 + 3 = 5 candies, which is the greatest among the
kids.
- Kid 2, they will have 3 + 3 = 6 candies, which is the greatest among the
kids.
- Kid 3, they will have 5 + 3 = 8 candies, which is the greatest among the
kids.
- Kid 4, they will have 1 + 3 = 4 candies, which is not the greatest among
the kids.
- Kid 5, they will have 3 + 3 = 6 candies, which is the greatest among the
kids.
```

Example 2:

```
Input: candies = [4,2,1,1,2], extraCandies = 1
Output: [true,false,false,false,false]
Explanation: There is only 1 extra candy.
Kid 1 will always have the greatest number of candies, even if a different
kid is given the extra candy.
```

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {
        List<Boolean> result = new ArrayList<>();

        int maxCandies = 0;
        for (int candy : candies) {
            maxCandies = Math.max(maxCandies, candy);
        }

        for (int candy : candies) {
            result.add(candy + extraCandies >= maxCandies);
        }

        return result;
    }
}
```

Description | Editorial | Solutions | Submissions

605. Can Place Flowers

Solved

Easy | Topics | Companies

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in **adjacent** plots.

Given an integer array `flowerbed` containing `0`'s and `1`'s, where `0` means empty and `1` means not empty, and an integer `n`, return `true` *if* `n` *new flowers can be planted in the* `flowerbed` *without violating the no-adjacent-flowers rule and* `false` *otherwise.*

Example 1:

```
Input: flowerbed = [1,0,0,0,1], n = 1
Output: true
```

Example 2:

```
Input: flowerbed = [1,0,0,0,1], n = 2
Output: false
```

Constraints:

- `1 <= flowerbed.length <= 2 * 10^4`
- `flowerbed[i]` is `0` or `1`.
- There are no two adjacent flowers in `flowerbed`.
- `0 <= n <= flowerbed.length`

Code

Java | Auto

```java
class Solution {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
        if (n == 0) {
            return true;
        }

        for (int i = 0; i < flowerbed.length; i++) {
            if (flowerbed[i] == 0 &&
                (i == 0 || flowerbed[i - 1] == 0) &&
                (i == flowerbed.length - 1 || flowerbed[i + 1] == 0)) {

                flowerbed[i] = 1;

                if (--n == 0) {
                    return true;
                }
            }
        }

        return false;
    }
}
```

Description | Editorial | Solutions | Submissions

345. Reverse Vowels of a String

Solved

Easy | Topics | Companies

Given a string `s`, reverse only all the vowels in the string and return it.

The vowels are `'a'`, `'e'`, `'i'`, `'o'`, and `'u'`, and they can appear in both lower and upper cases, more than once.

Example 1:

```
Input: s = "IceCreAm"

Output: "AceCreIm"

Explanation:

The vowels in s are ['I', 'e', 'e', 'A']. On reversing the vowels, s becomes "AceCreIm".
```

Example 2:

```
Input: s = "leetcode"

Output: "leotcede"
```

Constraints:

- `1 <= s.length <= 3 * 10^5`
- `s` consist of **printable ASCII** characters.

Code

Java | Auto

```java
import java.util.HashSet;

public class Solution {
    public String reverseVowels(String s) {
        char[] chars = s.toCharArray();
        HashSet<Character> vowels = new HashSet<>();
        vowels.addAll(Set.of('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'));

        int left = 0, right = chars.length - 1;
        while (left < right) {
            while (left < right && !vowels.contains(chars[left])) {
                left++;
            }
            while (left < right && !vowels.contains(chars[right])) {
                right--;
            }
            if (left < right) {
                char temp = chars[left];
                chars[left] = chars[right];
                chars[right] = temp;
                left++;
                right--;
            }
        }

        return new String(chars);
    }
}
```

Description | Editorial | Solutions | Submissions

151. Reverse Words in a String

Solved

Medium | Topics | Companies

Given an input string `s`, reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in `s` will be separated by at least one space.

Return *a string of the words in reverse order concatenated by a single space.*

Note that `s` may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

```
Input: s = "the sky is blue"
Output: "blue is sky the"
```

Example 2:

```
Input: s = "  hello world  "
Output: "world hello"
Explanation: Your reversed string should not contain leading or trailing
spaces.
```

Example 3:

```
Input: s = "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a
single space in the reversed string.
```

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public String reverseWords(String s) {
        s = s.trim(); // Remove leading and trailing spaces
        String[] words = s.split("\\s+"); // Split by spaces
        StringBuilder sb = new StringBuilder();

        for (int i = words.length - 1; i >= 0; i--) {
            sb.append(words[i]);
            if (i > 0) {
                sb.append(" ");
            }
        }

        return sb.toString();
    }
}
```

Description | Editorial | Solutions | Submissions

238. Product of Array Except Self

Solved

Medium | Topics | Companies | Hint

Given an integer array `nums`, return *an array* `answer` *such that* `answer[i]` *is equal to the product of all the elements of* `nums` *except* `nums[i]`.

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in `O(n)` time and without using the division operation.

Example 1:

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

Example 2:

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

Constraints:

- `2 <= nums.length <= 10^5`
- `-30 <= nums[i] <= 30`
- The input is generated such that `answer[i]` is **guaranteed** to fit in a **32-bit** integer.

Follow up: Can you solve the problem in `O(1)` extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

Code

Java | Auto

```java
public class Solution {
    public int[] productExceptSelf(int[] nums) {
        int n = nums.length;

        int[] answer = new int[n];

        for (int i = 0; i < n; i++) {
            answer[i] = 1;
        }

        int prefix = 1;
        for (int i = 0; i < n; i++) {
            answer[i] = prefix;
            prefix *= nums[i];
        }

        int postfix = 1;
        for (int i = n - 1; i >= 0; i--) {
            answer[i] *= postfix;
            postfix *= nums[i];
        }

        return answer;
    }
}
```

Description | Editorial | Solutions | Submissions

334. Increasing Triplet Subsequence

Solved

Medium | Topics | Companies

Given an integer array `nums`, return `true` *if there exists a triple of indices* `(i, j, k)` *such that* `i < j < k` *and* `nums[i] < nums[j] < nums[k]`. If no such indices exists, return `false`.

Example 1:

```
Input: nums = [1,2,3,4,5]
Output: true
Explanation: Any triplet where i < j < k is valid.
```

Example 2:

```
Input: nums = [5,4,3,2,1]
Output: false
Explanation: No triplet exists.
```

Example 3:

```
Input: nums = [2,1,5,0,4,6]
Output: true
Explanation: The triplet (3, 4, 5) is valid because nums[3] == 0 < nums[4]
== 4 < nums[5] == 6.
```

Constraints:

- `1 <= nums.length <= 5 * 10^5`
- `-2^31 <= nums[i] <= 2^31 - 1`

Follow up: Could you implement a solution that runs in `O(n)` time complexity and `O(1)` space complexity?

Code

Java | Auto

```java
class Solution {
    public boolean increasingTriplet(int[] nums) {
        int first = Integer.MAX_VALUE;
        int second = Integer.MAX_VALUE;

        for (int num : nums) {
            if (num <= first) {
                first = num;
            } else if (num <= second) {
                second = num;
            } else {
                return true; // Found a third number greater than both first and second
            }
        }

        return false;
    }
}
```

Description | Editorial | Solutions | Submissions

443. String Compression

Solved

Medium | Topics | Companies | Hint

Given an array of characters `chars`, compress it using the following algorithm:

Begin with an empty string `s`. For each group of **consecutive repeating characters** in `chars`:

- If the group's length is `1`, append the character to `s`.
- Otherwise, append the character followed by the group's length.

The compressed string `s` **should not be returned separately**, but instead, be stored **in the input character array** `chars`. Note that group lengths that are `10` or longer will be split into multiple characters in `chars`.

After you are done **modifying the input array,** return *the new length of the array*.

You must write an algorithm that uses only constant extra space.

Example 1:

```
Input: chars = ["a","a","b","b","c","c","c"]
Output: Return 6, and the first 6 characters of the input array should be:
["a","2","b","2","c","3"]
Explanation: The groups are "aa", "bb", and "ccc". This compresses to
"a2b2c3".
```

Example 2:

```
Input: chars = ["a"]
Output: Return 1, and the first character of the input array should be:
["a"]
Explanation: The only group is "a", which remains uncompressed since it's a
single character.
```

Code

Java | Auto

```java
class Solution {
    public int compress(char[] chars) {
        int write = 0, read = 0;

        while (read < chars.length) {
            char currentChar = chars[read];
            int count = 0;

            while (read < chars.length && chars[read] == currentChar) {
                read++;
                count++;
            }

            chars[write++] = currentChar;

            if (count > 1) {
                for (char c : Integer.toString(count).toCharArray()) {
                    chars[write++] = c;
                }
            }
        }

        return write;
    }
}
```

Two Pointers

Description | Editorial | Solutions | Submissions

283. Move Zeroes

Solved

Easy | Topics | Companies | Hint

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

Constraints:

- `1 <= nums.length <= 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`

Code

Java | Auto

```java
public class Solution {
    public void moveZeroes(int[] nums) {
        int slow = 0;

        for (int fast = 0; fast < nums.length; fast++) {
            if (nums[fast] != 0) {
                int temp = nums[slow];
                nums[slow] = nums[fast];
                nums[fast] = temp;
                slow++;
            }
        }
    }
}
```

Description | Editorial | Solutions | Submissions

392. Is Subsequence

Solved

Easy | Topics | Companies

Given two strings `s` and `t`, return `true` *if* `s` *is a* ***subsequence*** *of* `t`, *or* `false` *otherwise*.

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., `"ace"` is a subsequence of `"abcde"` while `"aec"` is not).

Example 1:

```
Input: s = "abc", t = "ahbgdc"
Output: true
```

Example 2:

```
Input: s = "axc", t = "ahbgdc"
Output: false
```

Constraints:

- `0 <= s.length <= 100`
- `0 <= t.length <= 10^4`
- `s` and `t` consist only of lowercase English letters.

Follow up: Suppose there are lots of incoming `s`, say `s1, s2, ..., sk` where `k >= 10^9`, and you want to check one by one to see if `t` has its subsequence. In this scenario, how would you change your code?

Code

Java | Auto

```java
class Solution {
    public boolean isSubsequence(String s, String t) {
        int i = 0, j = 0;
        while (i < s.length() && j < t.length()) {
            if (s.charAt(i) == t.charAt(j)) {
                i++;
            }
            j++;
        }
        return i == s.length();
    }
}
```

Description | Editorial | Solutions | Submissions

11. Container With Most Water

Solved

Medium | Topics | Companies | Hint

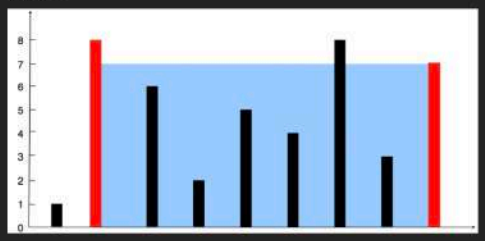
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the ith line are `(i, 0)` and `(i, height[i])`.

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.
```

Code

Java | Auto

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

        while (left < right) {
            int currentArea = Math.min(height[left], height[right]) * (right - left);
            max = Math.max(max, currentArea);

            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }

        return max;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] heights = {1, 2, 1};
        System.out.println(solution.maxArea(heights)); // Output: 2
    }
}
```

Description | Editorial | Solutions | Submissions

1679. Max Number of K-Sum Pairs

Solved

Medium | Topics | Companies | Hint

You are given an integer array `nums` and an integer `k`.

In one operation, you can pick two numbers from the array whose sum equals `k` and remove them from the array.

Return *the maximum number of operations you can perform on the array.*

Example 1:

```
Input: nums = [1,2,3,4], k = 5
Output: 2
Explanation: Starting with nums = [1,2,3,4]:
- Remove numbers 1 and 4, then nums = [2,3]
- Remove numbers 2 and 3, then nums = []
There are no more pairs that sum up to 5, hence a total of 2 operations.
```

Example 2:

```
Input: nums = [3,1,3,4,3], k = 6
Output: 1
Explanation: Starting with nums = [3,1,3,4,3]:
- Remove the first two 3's, then nums = [1,4,3]
There are no more pairs that sum up to 6, hence a total of 1 operation.
```

Constraints:

- `1 <= nums.length <= 10^5`
- `1 <= nums[i] <= 10^9`
- `1 <= k <= 10^9`

Code

Java | Auto

```java
import java.util.HashMap;
import java.util.Map;

class Solution {
    public int maxOperations(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        int count = 0;

        for (int num : nums) {
            int complement = k - num;

            if (map.getOrDefault(complement, 0) > 0) {
                count++;
                map.put(complement, map.get(complement) - 1);
            } else {
                map.put(num, map.getOrDefault(num, 0) + 1);
            }
        }

        return count;
    }
}
```

Sliding Window

Description | Editorial | Solutions | Submissions

643. Maximum Average Subarray I

Solved

Easy | Topics | Companies

You are given an integer array `nums` consisting of `n` elements, and an integer `k`.

Find a contiguous subarray whose **length is equal to** `k` that has the maximum average value and return *this value*. Any answer with a calculation error less than `$10^{-5}$` will be accepted.

Example 1:

```
Input: nums = [1,12,-5,-6,50,3], k = 4
Output: 12.75000
Explanation: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75
```

Example 2:

```
Input: nums = [5], k = 1
Output: 5.00000
```

Constraints:

- `n == nums.length`
- `$1 <= k <= n <= 10^5$`
- `$-10^4 <= nums[i] <= 10^4$`

Code

Java | Auto

```java
public class Solution {
    public double findMaxAverage(int[] nums, int k) {
        int sum = 0;
        for (int i = 0; i < k; ++i) {
            sum += nums[i];
        }

        double res = sum;

        for (int i = k; i < nums.length; ++i) {
            sum += nums[i] - nums[i - k];
            res = Math.max(res, sum);
        }

        return res / k;
    }
}
```

Description | Editorial | Solutions | Submissions

1456. Maximum Number of Vowels in a Substring of Given Length

Solved

Medium | Topics | Companies | Hint

Given a string `s` and an integer `k`, return *the maximum number of vowel letters in any substring of* `s` *with length* `k`.

Vowel letters in English are `'a'`, `'e'`, `'i'`, `'o'`, and `'u'`.

Example 1:

```
Input: s = "abciiidef", k = 3
Output: 3
Explanation: The substring "iii" contains 3 vowel letters.
```

Example 2:

```
Input: s = "aeiou", k = 2
Output: 2
Explanation: Any substring of length 2 contains 2 vowels.
```

Example 3:

```
Input: s = "leetcode", k = 3
Output: 2
Explanation: "lee", "eet" and "ode" contain 2 vowels.
```

Constraints:

- `$1 <= s.length <= 10^5$`
- `s` consists of lowercase English letters.
- `1 <= k <= s.length`

Code

Java | Auto

```java
class Solution {
    public int maxVowels(String s, int k) {
        int maxVowels = 0;
        int currentVowels = 0;
        int n = s.length();

        for (int i = 0; i < k; i++) {
            if (isVowel(s.charAt(i))) {
                currentVowels++;
            }
        }

        maxVowels = currentVowels;

        for (int i = k; i < n; i++) {
            if (isVowel(s.charAt(i))) {
                currentVowels++;
            }
            if (isVowel(s.charAt(i - k))) {
                currentVowels--;
            }
            maxVowels = Math.max(maxVowels, currentVowels);
        }

        return maxVowels;
    }

    private boolean isVowel(char c) {
        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||
               c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U';
    }
}
```

Description | Editorial | Solutions | Submissions

1004. Max Consecutive Ones III

Solved

Medium | Topics | Companies | Hint

Given a binary array `nums` and an integer `k`, return *the maximum number of consecutive* `1`*'s in the array if you can flip at most* `k` `0`'s.

Example 1:

```
Input: nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2
Output: 6
Explanation: [1,1,1,0,0,1,1,1,1,1,1]
Bolded numbers were flipped from 0 to 1. The longest subarray is
underlined.
```

Example 2:

```
Input: nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], k = 3
Output: 10
Explanation: [0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
Bolded numbers were flipped from 0 to 1. The longest subarray is
underlined.
```

Constraints:

- `1 <= nums.length <= 10^5`
- `nums[i]` is either `0` or `1`.
- `0 <= k <= nums.length`

Code

Java | Auto

```java
public class Solution {
    public int longestOnes(int[] nums, int k) {
        int left = 0;
        int right = 0;
        int maxOnes = 0;
        int zerosCount = 0;

        for (right = 0; right < nums.length; right++) {
            if (nums[right] == 0) {
                zerosCount++;
            }

            while (zerosCount > k) {
                if (nums[left] == 0) {
                    zerosCount--;
                }
                left++;
            }

            maxOnes = Math.max(maxOnes, right - left + 1);
        }

        return maxOnes;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums = {1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0};
        int k = 2;
        System.out.println(solution.longestOnes(nums, k)); // Output: 6
    }
}
```

Description | Editorial | Solutions | Submissions

1493. Longest Subarray of 1's After Deleting One Element

Solved

Medium | Topics | Companies | Hint

Given a binary array `nums`, you should delete one element from it.

Return *the size of the longest non-empty subarray containing only* `1`*'s in the resulting array*. Return `0` if there is no such subarray.

Example 1:

```
Input: nums = [1,1,0,1]
Output: 3
Explanation: After deleting the number in position 2, [1,1,1] contains 3
numbers with value of 1's.
```

Example 2:

```
Input: nums = [0,1,1,1,0,1,1,0,1]
Output: 5
Explanation: After deleting the number in position 4, [0,1,1,1,1,1,0,1]
longest subarray with value of 1's is [1,1,1,1,1].
```

Example 3:

```
Input: nums = [1,1,1]
Output: 2
Explanation: You must delete one element.
```

Constraints:

- `1 <= nums.length <= 10^5`
- `nums[i]` is either `0` or `1`.

Code

Java | Auto

```java
class Solution {
    public int longestSubarray(int[] nums) {
        int left = 0;
        int zeroCount = 0;
        int maxLength = 0;

        for (int right = 0; right < nums.length; right++) {
            if (nums[right] == 0) {
                zeroCount++;
            }

            while (zeroCount > 1) {
                if (nums[left] == 0) {
                    zeroCount--;
                }
                left++;
            }

            maxLength = Math.max(maxLength, right - left);
        }

        return maxLength;
    }
}
```

Prefix Sum

Description | Editorial | Solutions | Submissions

1732. Find the Highest Altitude

Solved

Easy | Topics | Companies | Hint

There is a biker going on a road trip. The road trip consists of `n + 1` points at different altitudes. The biker starts his trip on point `0` with altitude equal `0`.

You are given an integer array `gain` of length `n` where `gain[i]` is the **net gain in altitude** between points `i` and `i + 1` for all (`0 <= i < n`). Return *the **highest altitude** of a point.*

Example 1:

```
Input: gain = [-5,1,5,0,-7]
Output: 1
Explanation: The altitudes are [0,-5,-4,1,1,-6]. The highest is 1.
```

Example 2:

```
Input: gain = [-4,-3,-2,-1,4,3,2]
Output: 0
Explanation: The altitudes are [0,-4,-7,-9,-10,-6,-3,-1]. The highest is 0.
```

Constraints:

- `n == gain.length`
- `1 <= n <= 100`
- `-100 <= gain[i] <= 100`

Code

Java | Auto

```java
class Solution {
    public int largestAltitude(int[] gain) {
        int altitude = 0; // starting point
        int maxAltitude = 0;

        for (int g : gain) {
            altitude += g;
            maxAltitude = Math.max(maxAltitude, altitude);
        }

        return maxAltitude;
    }
}
```

Description | Editorial | Solutions | Submissions

724. Find Pivot Index

Solved

Easy | Topics | Companies | Hint

Given an array of integers `nums`, calculate the **pivot index** of this array.

The **pivot index** is the index where the sum of all the numbers **strictly** to the left of the index is equal to the sum of all the numbers **strictly** to the index's right.

If the index is on the left edge of the array, then the left sum is `0` because there are no elements to the left. This also applies to the right edge of the array.

Return *the **leftmost pivot index**.* If no such index exists, return `-1`.

Example 1:

```
Input: nums = [1,7,3,6,5,6]
Output: 3
Explanation:
The pivot index is 3.
Left sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11
Right sum = nums[4] + nums[5] = 5 + 6 = 11
```

Example 2:

```
Input: nums = [1,2,3]
Output: -1
Explanation:
There is no index that satisfies the conditions in the problem statement.
```

Example 3:

```
Input: nums = [2,1,-1]
Output: 0
Explanation:
The pivot index is 0.
Left sum = 0 (no elements to the left of index 0)
Right sum = nums[1] + nums[2] = 1 + -1 = 0
```

Code

Java | Auto

```java
public class Solution {
    public int pivotIndex(int[] nums) {
        int totalSum = 0;
        for (int num : nums) {
            totalSum += num;
        }

        int leftSum = 0;

        for (int i = 0; i < nums.length; i++) {
            if (leftSum == totalSum - leftSum - nums[i]) {
                return i;
            }

            leftSum += nums[i];
        }

        return -1;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums = {1, 7, 3, 6, 5, 6};
        System.out.println(solution.pivotIndex(nums)); // Output: 3
    }
}
```

HashMap / Set

Description | Editorial | Solutions | Submissions

2215. Find the Difference of Two Arrays

Solved

Easy | Topics | Companies | Hint

Given two **0-indexed** integer arrays `nums1` and `nums2`, return *a list* `answer` *of size* `2` *where:*

- `answer[0]` *is a list of all* ***distinct*** *integers in* `nums1` *which are* ***not*** *present in* `nums2`.
- `answer[1]` *is a list of all* ***distinct*** *integers in* `nums2` *which are* ***not*** *present in* `nums1`.

Note that the integers in the lists may be returned in **any** order.

Example 1:

```
Input: nums1 = [1,2,3], nums2 = [2,4,6]
Output: [[1,3],[4,6]]
Explanation:
For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] =
1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].
For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] =
4 and nums2[2] = 6 are not present in nums1. Therefore, answer[1] = [4,6].
```

Example 2:

```
Input: nums1 = [1,2,3,3], nums2 = [1,1,2,2]
Output: [[3],[]]
Explanation:
For nums1, nums1[2] and nums1[3] are not present in nums2. Since nums1[2]
== nums1[3], their value is only included once and answer[0] = [3].
Every integer in nums2 is present in nums1. Therefore, answer[1] = [].
```

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`
- `-1000 <= nums1[i], nums2[i] <= 1000`

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public List<List<Integer>> findDifference(int[] nums1, int[] nums2) {
        Set<Integer> set1 = new HashSet<>();
        Set<Integer> set2 = new HashSet<>();

        for (int num : nums1) {
            set1.add(num);
        }
        for (int num : nums2) {
            set2.add(num);
        }

        List<Integer> diff1 = new ArrayList<>();
        for (int num : set1) {
            if (!set2.contains(num)) {
                diff1.add(num);
            }
        }

        List<Integer> diff2 = new ArrayList<>();
        for (int num : set2) {
            if (!set1.contains(num)) {
                diff2.add(num);
            }
        }

        List<List<Integer>> result = new ArrayList<>();
        result.add(diff1);
        result.add(diff2);

        return result;
    }
}
```

Description | Editorial | Solutions | Submissions

1207. Unique Number of Occurrences

Solved

Easy | Topics | Companies | Hint

Given an array of integers `arr`, return `true` *if the number of occurrences of each value in the array is* ***unique*** *or* `false` *otherwise.*

Example 1:

```
Input: arr = [1,2,2,1,1,3]
Output: true
Explanation: The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two
values have the same number of occurrences.
```

Example 2:

```
Input: arr = [1,2]
Output: false
```

Example 3:

```
Input: arr = [-3,0,1,-3,1,1,1,-3,10,0]
Output: true
```

Constraints:

- `1 <= arr.length <= 1000`
- `-1000 <= arr[i] <= 1000`

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public boolean uniqueOccurrences(int[] arr) {
        Map<Integer, Integer> frequencyMap = new HashMap<>();

        for (int num : arr) {
            frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
        }

        Set<Integer> occurrencesSet = new HashSet<>();

        for (int freq : frequencyMap.values()) {
            if (occurrencesSet.contains(freq)) {
                return false; // duplicate frequency found
            }
            occurrencesSet.add(freq);
        }

        return true;
    }
}
```

Description | Editorial | Solutions | Submissions

1657. Determine if Two Strings Are Close

Solved

Medium | Topics | Companies | Hint

Two strings are considered **close** if you can attain one from the other using the following operations:

- Operation 1: Swap any two **existing** characters.
  - For example, `abcde -> aecdb`
- Operation 2: Transform **every** occurrence of one **existing** character into another **existing** character, and do the same with the other character.
  - For example, `aacabb -> bbcbaa` (all `a`'s turn into `b`'s, and all `b`'s turn into `a`'s)

You can use the operations on either string as many times as necessary.

Given two strings, `word1` and `word2`, return `true` *if* `word1` *and* `word2` *are* ***close****, and* `false` *otherwise.*

Example 1:

```
Input: word1 = "abc", word2 = "bca"
Output: true
Explanation: You can attain word2 from word1 in 2 operations.
Apply Operation 1: "abc" -> "acb"
Apply Operation 1: "acb" -> "bca"
```

Example 2:

```
Input: word1 = "a", word2 = "aa"
Output: false
Explanation: It is impossible to attain word2 from word1, or vice versa, in
any number of operations.
```

Code

Java | Auto

```java
import java.util.Arrays;

class Solution {
    public boolean closeStrings(String word1, String word2) {
        if (word1.length() != word2.length()) {
            return false;
        }

        int[] freq1 = new int[26];
        int[] freq2 = new int[26];

        for (char c : word1.toCharArray()) {
            freq1[c - 'a']++;
        }
        for (char c : word2.toCharArray()) {
            freq2[c - 'a']++;
        }

        for (int i = 0; i < 26; i++) {
            if ((freq1[i] == 0 && freq2[i] != 0) ||
                (freq1[i] != 0 && freq2[i] == 0)) {
                return false;
            }
        }

        Arrays.sort(freq1);
        Arrays.sort(freq2);

        return Arrays.equals(freq1, freq2);
    }
}
```

Description | Editorial | Solutions | Submissions

2352. Equal Row and Column Pairs

Solved

Medium | Topics | Companies | Hint

Given a **0-indexed** `n x n` integer matrix `grid`, *return the number of pairs* (r_i, c_j) *such that row* r_i *and column* c_j *are equal.*

A row and column pair is considered equal if they contain the same elements in the same order (i.e., an equal array).

Example 1:

3	2	1
1	7	6
2	7	7

```
Input: grid = [[3,2,1],[1,7,6],[2,7,7]]
Output: 1
Explanation: There is 1 equal row and column pair:
- (Row 2, Column 1): [2,7,7]
```

Example 2:

3	1	2	2
1	4	4	5
2	4	2	2
2	4	2	2

```
Input: grid = [[3,1,2,2],[1,4,4,5],[2,4,2,2],[2,4,2,2]]
Output: 3
```

Code

Java | Auto

```java
class Solution {
    private static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    private int n;

    public int equalPairs(int[][] grid) {
        n = grid.length;
        int count = 0;

        for (int row = 0; row < n; row++) {
            for (int col = 0; col < n; col++) {
                if (checkEquality(grid, row, col)) {
                    count++;
                }
            }
        }

        return count;
    }

    private boolean checkEquality(int[][] grid, int row, int col) {
        for (int i = 0; i < n; i++) {
            if (grid[row][i] != grid[i][col]) {
                return false;
            }
        }
        return true;
    }
}
```

Saved

Stack

Description | Editorial | Solutions | Submissions

2390. Removing Stars From a String

Solved

Medium | Topics | Companies | Hint

You are given a string `s`, which contains stars `*`.

In one operation, you can:

- Choose a star in `s`.
- Remove the closest **non-star** character to its **left**, as well as remove the star itself.

Return *the string after **all** stars have been removed.*

Note:

- The input will be generated such that the operation is always possible.
- It can be shown that the resulting string will always be unique.

Example 1:

```
Input: s = "leet**cod*e"
Output: "lecoe"
Explanation: Performing the removals from left to right:
- The closest character to the 1st star is 't' in "leet**cod*e". s becomes
"lee*cod*e".
- The closest character to the 2nd star is 'e' in "lee*cod*e". s becomes
"lecod*e".
- The closest character to the 3rd star is 'd' in "lecod*e". s becomes
"lecoe".
There are no more stars, so we return "lecoe".
```

Example 2:

```
Input: s = "erase*****"
Output: ""
Explanation: The entire string is removed, so we return an empty string.
```

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public String removeStars(String s) {
        Deque<Integer> stack = new ArrayDeque<>();

        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (ch == '*') {
                if (!stack.isEmpty()) {
                    stack.pop(); // Remove closest non-star character on the left
                }
            } else {
                stack.push(i); // Push index of non-star character
            }
        }

        StringBuilder result = new StringBuilder();
        while (!stack.isEmpty()) {
            result.append(s.charAt(stack.pop()));
        }

        return result.reverse().toString();
    }
}
```

Description | Editorial | Solutions | Submissions

735. Asteroid Collision

Solved

Medium | Topics | Companies | Hint

We are given an array `asteroids` of integers representing asteroids in a row. The indices of the asteriod in the array represent their relative position in space.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:

```
Input: asteroids = [5,10,-5]
Output: [5,10]
Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never
collide.
```

Example 2:

```
Input: asteroids = [8,-8]
Output: []
Explanation: The 8 and -8 collide exploding each other.
```

Example 3:

```
Input: asteroids = [10,2,-5]
Output: [10]
Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide
resulting in 10.
```

Code

Java | Auto

```java
// [5,10,-5]
class Solution {
    public int[] asteroidCollision(int[] asteroids) {
        Stack<Integer> stack = new Stack<>();
        for (final int a : asteroids)
            if (a > 0) {
                stack.push(a);
            } else {
                while (!stack.isEmpty() && stack.peek() > 0 && stack.peek() < -a)
                    stack.pop();
                if (stack.isEmpty() || stack.peek() < 0)
                    stack.push(a);
                else if (stack.peek() == -a)
                    stack.pop();
            }
        int[] ans = new int[stack.size()];
        for (int i = ans.length - 1; i >= 0; --i)
            ans[i] = stack.pop();
        return ans;
    }
}
```

Description | Editorial | Solutions | Submissions

394. Decode String

Solved

Medium | Topics | Companies

Given an encoded string, return its decoded string.

The encoding rule is: `k[encoded_string]`, where the `encoded_string` inside the square brackets is being repeated exactly `k` times. Note that `k` is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, `k`. For example, there will not be input like `3a` or `2[4]`.

The test cases are generated so that the length of the output will never exceed 10^5.

Example 1:

```
Input: s = "3[a]2[bc]"
Output: "aaabcbc"
```

Example 2:

```
Input: s = "3[a2[c]]"
Output: "accaccacc"
```

Example 3:

```
Input: s = "2[abc]3[cd]ef"
Output: "abcabccdcdcdef"
```

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public String decodeString(String s) {
        Stack<Integer> countStack = new Stack<>();
        Stack<StringBuilder> stringStack = new Stack<>();
        StringBuilder currentStr = new StringBuilder();
        int num = 0;

        for (char ch : s.toCharArray()) {
            if (Character.isDigit(ch)) {
                num = num * 10 + (ch - '0');
            } else if (ch == '[') {
                countStack.push(num);
                stringStack.push(currentStr);
                currentStr = new StringBuilder();
                num = 0;
            } else if (ch == ']') {
                int count = countStack.pop();
                StringBuilder decodedStr = stringStack.pop();
                while (count-- > 0) {
                    decodedStr.append(currentStr);
                }
                currentStr = decodedStr;
            } else {
                currentStr.append(ch);
            }
        }

        return currentStr.toString();
    }
}
```

Queue

Description | Editorial | Solutions | Submissions

933. Number of Recent Calls

Solved

Easy | Topics | Companies

You have a `RecentCounter` class which counts the number of recent requests within a certain time frame.

Implement the `RecentCounter` class:

- `RecentCounter()` Initializes the counter with zero recent requests.
- `int ping(int t)` Adds a new request at time `t`, where `t` represents some time in milliseconds, and returns the number of requests that has happened in the past `3000` milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range `[t - 3000, t]`.

It is **guaranteed** that every call to `ping` uses a strictly larger value of `t` than the previous call.

Example 1:

```
Input
["RecentCounter", "ping", "ping", "ping", "ping"]
[[], [1], [100], [3001], [3002]]
Output
[null, 1, 2, 3, 3]

Explanation
RecentCounter recentCounter = new RecentCounter();
recentCounter.ping(1);     // requests = [1], range is [-2999,1], return 1
recentCounter.ping(100);   // requests = [1, 100], range is [-2900,100],
return 2
recentCounter.ping(3001);  // requests = [1, 100, 3001], range is [1,3001],
return 3
recentCounter.ping(3002);  // requests = [1, 100, 3001, 3002], range is
[2,3002], return 3
```

Code

Java | Auto

```java
class RecentCounter {
    private Queue<Integer> recentPings;

    public RecentCounter() {
        recentPings = new LinkedList<>();
    }

    public int ping(int t) {
        recentPings.offer(t);
        while (recentPings.peek() < t - 3000) {
            recentPings.poll();  // Remove outdated pings
        }
        return recentPings.size();  // Number of valid pings in the window
    }
}


/**
 * Your RecentCounter object will be instantiated and called as such:
 * RecentCounter obj = new RecentCounter();
 * int param_1 = obj.ping(t);
 */

```

Description | Editorial | Solutions | Submissions

649. Dota2 Senate

Solved

Medium | Topics | Companies

In the world of Dota2, there are two parties: the Radiant and the Dire.

The Dota2 senate consists of senators coming from two parties. Now the Senate wants to decide on a change in the Dota2 game. The voting for this change is a round-based procedure. In each round, each senator can exercise **one** of the two rights:

- **Ban one senator's right:** A senator can make another senator lose all his rights in this and all the following rounds.
- **Announce the victory:** If this senator found the senators who still have rights to vote are all from the same party, he can announce the victory and decide on the change in the game.

Given a string `senate` representing each senator's party belonging. The character `'R'` and `'D'` represent the Radiant party and the Dire party. Then if there are `n` senators, the size of the given string will be `n`.

The round-based procedure starts from the first senator to the last senator in the given order. This procedure will last until the end of voting. All the senators who have lost their rights will be skipped during the procedure.

Suppose every senator is smart enough and will play the best strategy for his own party. Predict which party will finally announce the victory and change the Dota2 game. The output should be `"Radiant"` or `"Dire"`.

Example 1:

```
Input: senate = "RD"
Output: "Radiant"
Explanation:
The first senator comes from Radiant and he can just ban the next senator's
right in round 1.
And the second senator can't exercise any rights anymore since his right
has been banned.
And in round 2, the first senator can just announce the victory since he is
the only guy in the senate who can vote.
```

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public String predictPartyVictory(String senate) {
        int n = senate.length();
        Queue<Integer> radiant = new LinkedList<>();
        Queue<Integer> dire = new LinkedList<>();

        // Fill queues with indices of senators
        for (int i = 0; i < n; i++) {
            if (senate.charAt(i) == 'R') {
                radiant.offer(i);
            } else {
                dire.offer(i);
            }
        }

        // Simulate rounds
        while (!radiant.isEmpty() && !dire.isEmpty()) {
            int rIndex = radiant.poll();
            int dIndex = dire.poll();

            if (rIndex < dIndex) {
                // Radiant bans Dire, Radiant senator goes to the next round
                radiant.offer(rIndex + n);
            } else {
                // Dire bans Radiant
                dire.offer(dIndex + n);
            }
        }

        return radiant.isEmpty() ? "Dire" : "Radiant";
    }
}
```

Linked List

Description | Editorial | Solutions | Submissions

2095. Delete the Middle Node of a Linked List

Solved

Medium | Topics | Companies | Hint

You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list.*

The **middle node** of a linked list of size `n` is the `⌊n / 2⌋`th node from the **start** using **0-based indexing**, where `⌊x⌋` denotes the largest integer less than or equal to `x`.

- For `n` = `1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

Example 1:


```
Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes
are written below.
Since n = 7, node 3 with value 7 is the middle node, which is marked in
red.
We return the new list after removing this node.
```

Example 2:


```
Input: head = [1,2,3,4]
Output: [1,2,4]
Explanation:
The above figure represents the given linked list.
For n = 4, node 2 with value 3 is the middle node, which is marked in red.
```

Code

Java | Auto

```java
/**
 * Definition for singly-linked list.
 * public 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 deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
            return null;
        }

        ListNode slow = head;
        ListNode fast = head;
        ListNode prev = null;

        while (fast != null && fast.next != null) {
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }

        prev.next = slow.next;

        return head;
    }
}
```

Description | Editorial | Solutions | Submissions

328. Odd Even Linked List

Solved

Medium | Topics | Companies

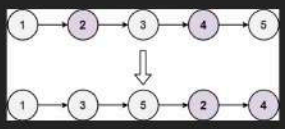
Given the `head` of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list.*

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

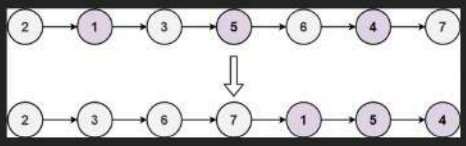
You must solve the problem in `O(1)` extra space complexity and `O(n)` time complexity.

Example 1:




```
Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]
```

Example 2:




```
Input: head = [2,1,3,5,6,4,7]
Output: [2,3,6,7,1,5,4]
```

Code

Java | Auto

```java
/**
 * Definitio.    .n for singly-linked list.
 * public 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 oddEvenList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        ListNode odd = head;
        ListNode even = head.next;
        ListNode evenHead = even;

        while (even != null && even.next != null) {
            odd.next = even.next;
            odd = odd.next;
            even.next = odd.next;
            even = even.next;
        }

        odd.next = evenHead;
        return head;
    }
}
```

Description | Editorial | Solutions | Submissions

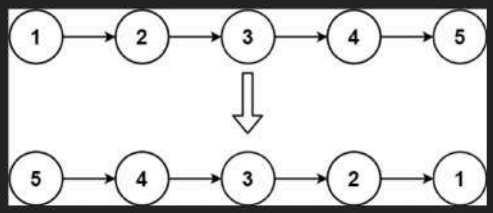
206. Reverse Linked List

Solved

Easy | Topics | Companies

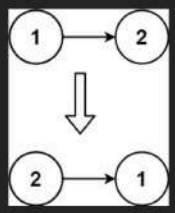
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:




```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

Example 2:




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

Code

Java | Auto

```java
/**
 * Definition for singly-linked list.
 * public 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 reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;

        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }

        return prev;
    }
}
```

Description | Editorial | Solutions | Submissions

2130. Maximum Twin Sum of a Linked List

Solved

Medium | Topics | Companies | Hint

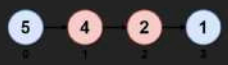
In a linked list of size `n`, where `n` is **even**, the `i`th node (**0-indexed**) of the linked list is known as the **twin** of the `(n-1-i)`th node, if `0 <= i <= (n / 2) - 1`.

- For example, if `n = 4`, then node `0` is the twin of node `3`, and node `1` is the twin of node `2`. These are the only nodes with twins for `n = 4`.

The **twin sum** is defined as the sum of a node and its twin.

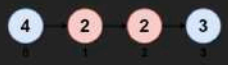
Given the `head` of a linked list with even length, return *the **maximum twin sum** of the linked list.*

Example 1:




```
Input: head = [5,4,2,1]
Output: 6
Explanation:
Nodes 0 and 1 are the twins of nodes 3 and 2, respectively. All have twin
sum = 6.
There are no other nodes with twins in the linked list.
Thus, the maximum twin sum of the linked list is 6.
```

Example 2:




```
Input: head = [4,2,2,3]
Output: 7
Explanation:
The nodes with twins present in this linked list are:
- Node 0 is the twin of node 3 having a twin sum of 4 + 3 = 7.
- Node 1 is the twin of node 2 having a twin sum of 2 + 2 = 4.
Thus, the maximum twin sum of the linked list is max(7, 4) = 7.
```

Code

Java | Auto

```java
/**
 * Definition for singly-linked list.
 * public 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 int pairSum(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        ListNode prev = null, curr = slow;
        while (curr != null) {
            ListNode next = curr.next;
            curr.next = prev;
            prev = curr;
            curr = next;
        }

        ListNode first = head;
        ListNode second = prev;
        int maxSum = 0;
        while (second != null) {
            maxSum = Math.max(maxSum, first.val + second.val);
            first = first.next;
            second = second.next;
        }

        return maxSum;
    }
}
```

Saved

Binary Tree - DFS

Description | Editorial | Solutions | Submissions

104. Maximum Depth of Binary Tree

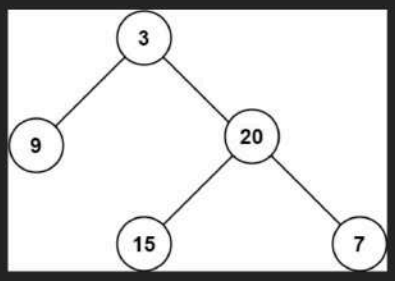
Solved

Easy | Topics | Companies

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

Example 2:

```
Input: root = [1,null,2]
Output: 2
```

Code

Java | Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);

        return 1 + Math.max(leftDepth, rightDepth);
    }
}
```

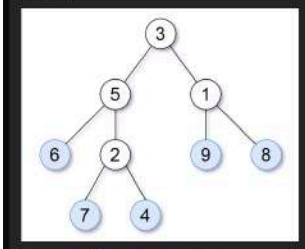
Description | Editorial | Solutions | Submissions

872. Leaf-Similar Trees

Solved

Easy | Topics | Companies

Consider all the leaves of a binary tree, from left to right order, the values of those leaves form a **leaf value sequence**.

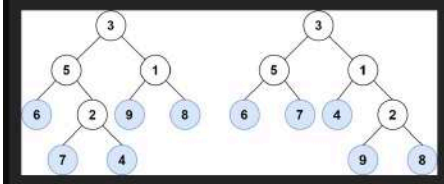


For example, in the given tree above, the leaf value sequence is `(6, 7, 4, 9, 8)`.

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

Return `true` if and only if the two given trees with head nodes `root1` and `root2` are leaf-similar.

Example 1:




```
Input: root1 = [3,5,1,6,2,9,8,null,null,7,4], root2 =
[3,5,1,6,7,4,2,null,null,null,null,null,null,9,8]
Output: true
```

Code

Java | Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean leafSimilar(TreeNode root1, TreeNode root2) {
        List<Integer> leaves1 = new ArrayList<>();
        List<Integer> leaves2 = new ArrayList<>();
        dfs(root1, leaves1);
        dfs(root2, leaves2);
        return leaves1.equals(leaves2);
    }

    public void dfs(TreeNode node, List<Integer> leaves) {
        if (node == null)
            return;
        if (node.left == null && node.right == null) {
            leaves.add(node.val);
            return;
        }

        dfs(node.left, leaves);
        dfs(node.right, leaves);
    }
}
```

Saved | Ln 1, Col 1

Description | Editorial | Solutions | Submissions

1448. Count Good Nodes in Binary Tree

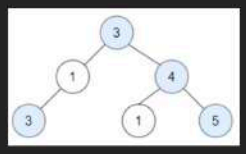
Solved

Medium | Topics | Companies | Hint

Given a binary tree `root`, a node *X* in the tree is named **good** if in the path from root to *X* there are no nodes with a value *greater than* X.

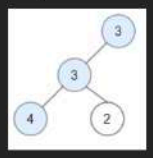
Return the number of **good** nodes in the binary tree.

Example 1:




```
Input: root = [3,1,4,3,null,1,5]
Output: 4
Explanation: Nodes in blue are good.
Root Node (3) is always a good node.
Node 4 -> (3,4) is the maximum value in the path starting from the root.
Node 5 -> (3,4,5) is the maximum value in the path
Node 3 -> (3,1,3) is the maximum value in the path.
```

Example 2:




```
Input: root = [3,3,null,4,2]
Output: 3
Explanation: Node 2 -> (3, 3, 2) is not good, because "3" is higher than
```

Code

Java | Auto

```java
/**
* Definition for a binary tree node.
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode() {}
*     TreeNode(int val) { this.val = val; }
*     TreeNode(int val, TreeNode left, TreeNode right) {
*         this.val = val;
*         this.left = left;
*         this.right = right;
*     }
* }
*/
class Solution {
    public int goodNodes(TreeNode root) {
        return dfs(root, root.val);
    }

    private int dfs(TreeNode node, int maxSoFar) {
        if (node == null) {
            return 0;
        }

        int count = 0;

        if (node.val >= maxSoFar) {
            count = 1;
        }

        maxSoFar = Math.max(maxSoFar, node.val);
        count += dfs(node.left, maxSoFar);
        count += dfs(node.right, maxSoFar);

        return count;
    }
}

```

Saved

Description | Editorial | Solutions | Submissions

437. Path Sum III

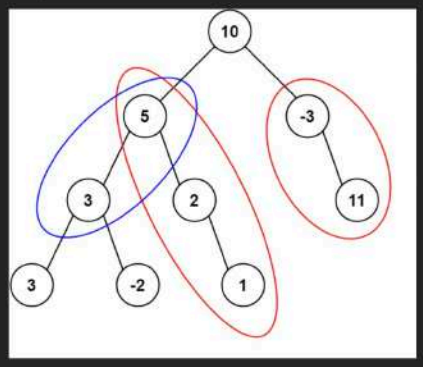
Solved

Medium | Topics | Companies

Given the `root` of a binary tree and an integer `targetSum`, return *the number of paths where the sum of the values along the path equals* `targetSum`.

The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).

Example 1:




```
Input: root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
Output: 3
Explanation: The paths that sum to 8 are shown.
```

Example 2:

```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: 3
```

Code

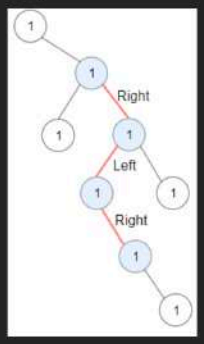
Java | Auto

```java
*         this.left = left;
*         this.right = right;
*     }
* }
*/
class Solution {
    public int pathSum(TreeNode root, int targetSum) {
        if (root == null) {
            return 0;
        }

        int count = countPathsFromNode(root, targetSum);
        count += pathSum(root.left, targetSum);
        count += pathSum(root.right, targetSum);

        return count;
    }

    private int countPathsFromNode(TreeNode node, long targetSum) {
        if (node == null) {
            return 0;
        }

        int count = 0;
        if (node.val == targetSum) {
            count++;
        }

        count += countPathsFromNode(node.left, targetSum - node.val);
        count += countPathsFromNode(node.right, targetSum - node.val);

        return count;
    }
}

```

Saved

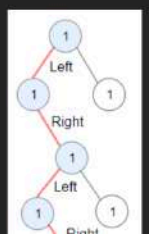
Description | Editorial | Solutions | Submissions

Example 1:




```
Input: root = [1,null,1,1,1,null,null,1,1,null,1,null,null,null,1]
Output: 3
Explanation: Longest ZigZag path in blue nodes (right -> left -> right).
```

Example 2:



Code

Java | Auto

```java
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    private int result = 0;

    public int longestZigZag(TreeNode root) {
        dfs(root, 0, true);  // Start going left
        dfs(root, 0, false); // Start going right
        return result;
    }

    private void dfs(TreeNode node, int length, boolean isLeft) {
        if (node == null) {
            return;
        }

        result = Math.max(result, length);

        if (isLeft) {
            dfs(node.left, length + 1, false); // continue zigzag
            dfs(node.right, 1, true);          // reset length when switching
        } else {
            dfs(node.right, length + 1, true);
            dfs(node.left, 1, false);
        }
    }
}
```

Saved

Description | Editorial | Solutions | Submissions

236. Lowest Common Ancestor of a Binary Tree

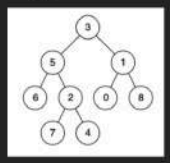
Solved

Medium | Topics | Companies

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

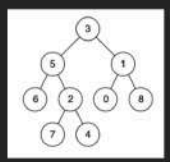
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Example 2:




```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
```

17.9K | 147 | 179 Online

Code

Java | Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null) {
            return null; // Base case
        }

        // If the current node is either p or q, return it
        if (root == p || root == q) {
            return root;
        }

        // Search in left and right subtrees
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);

        // If both left and right are non-null, this node is the LCA
        if (left != null && right != null) {
            return root;
        }

        // Otherwise, return whichever is non-null
        return (left != null) ? left : right;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

Binary Tree - BFS

LeetCode 75

Description | Editorial | Solutions | Submissions

199. Binary Tree Right Side View

Solved

Medium | Topics | Companies

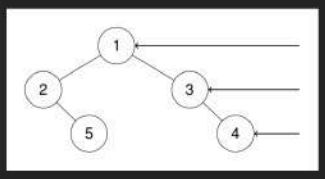
Given the `root` of a binary tree, imagine yourself standing on the **right side** of it, return *the values of the nodes you can see ordered from top to bottom.*

Example 1:

Input: `root = [1,2,3,null,5,null,4]`

Output: `[1,3,4]`

Explanation:

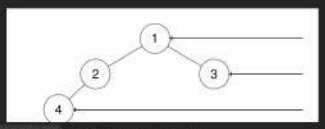


Example 2:

Input: `root = [1,2,3,4,null,null,null,5]`

Output: `[1,3,4,5]`

Explanation:



12.9K 277 196 Online

https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree?envType=study-plan-v2&envId=leetcode-75

Submit | Premium

Code

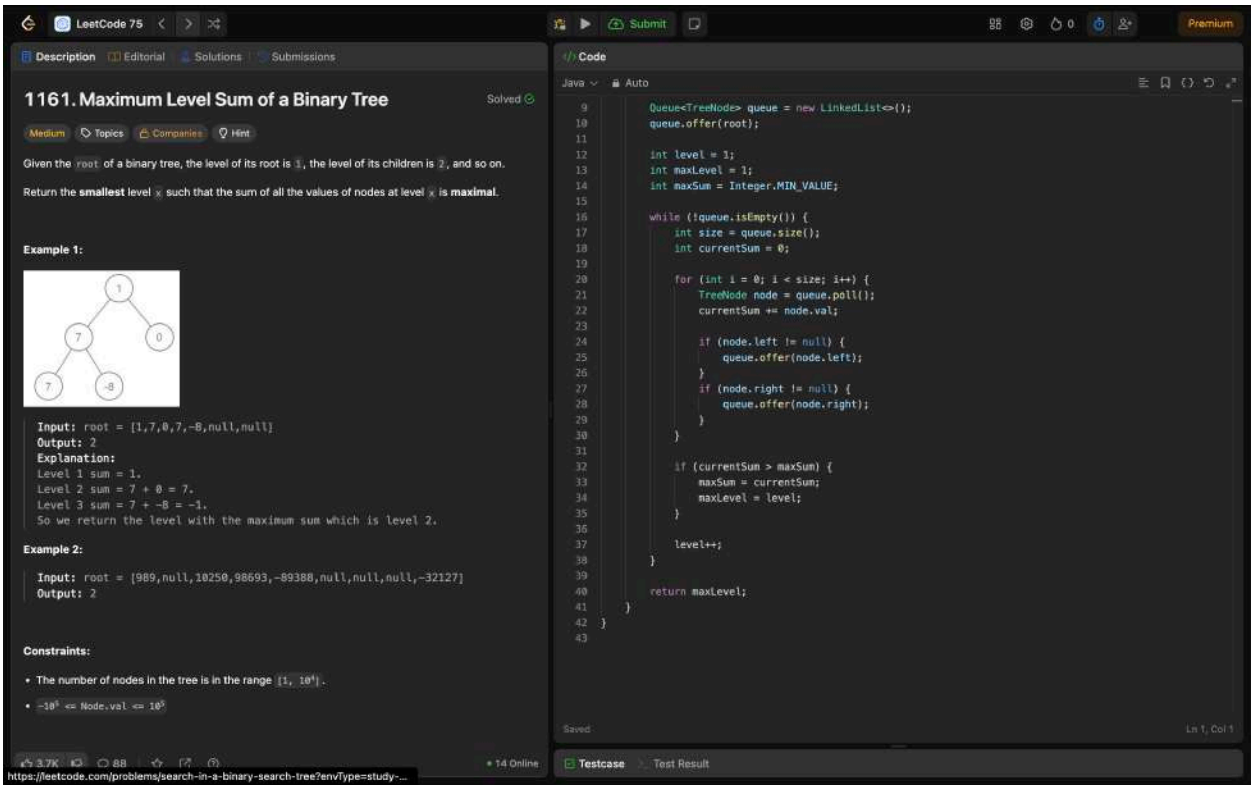
Java | Auto

```java
import java.util.*;

class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();

                // The last node in each level is the rightmost node
                if (i == size - 1) {
                    result.add(node.val);
                }

                if (node.left != null) {
                    queue.offer(node.left);
                }
                if (node.right != null) {
                    queue.offer(node.right);
                }
            }
        }

        return result;
    }
}
```

Saved

Ln 1, Col 1

Testcase | Test Result



Binary Search Tree

LeetCode 75

Description | Editorial | Solutions | Submissions

700. Search in a Binary Search Tree

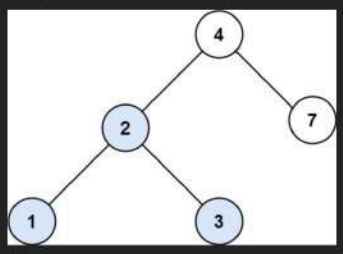
Solved

Easy | Topics | Companies

You are given the `root` of a binary search tree (BST) and an integer `val`.

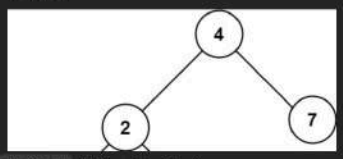
Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.

Example 1:




```
Input: root = [4,2,7,1,3], val = 2
Output: [2,1,3]
```

Example 2:



6.3K | 74 | 186 Online

Submit | Premium

Code

Java | Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode searchBST(TreeNode root, int val) {
        if (root == null || root.val == val) {
            return root;
        }

        if (val < root.val) {
            return searchBST(root.left, val);
        } else {
            return searchBST(root.right, val);
        }
    }
}
```

Saved | Ln 23, Col 46

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

450. Delete Node in a BST

Solved

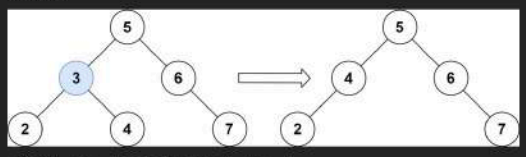
Medium | Topics | Companies

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return *the **root node reference** (possibly updated) of the BST.*

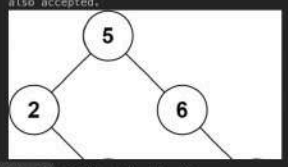
Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Example 1:




```
Input: root = [5,3,6,2,4,null,7], key = 3
Output: [5,4,6,2,null,null,7]
Explanation: Given key to delete is 3. So we find the node with value 3 and
delete it.
One valid answer is [5,4,6,2,null,null,7], shown in the above BST.
Please notice that another valid answer is [5,2,6,null,4,null,7] and it's
also accepted.
```



10K | 189 | 116 Online

Submit | Premium

Code

Java | Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 * int val;
 * TreeNode left;
 * TreeNode right;
 * TreeNode() {}
 * TreeNode(int val) { this.val = val; }
 * TreeNode(int val, TreeNode left, TreeNode right) {
 * this.val = val;
 * this.left = left;
 * this.right = right;
 * }
 * }
 */
class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if (root == null) {
            return null;
        }

        if (key < root.val) {
            root.left = deleteNode(root.left, key);
        } else if (key > root.val) {
            root.right = deleteNode(root.right, key);
        } else {
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            }
            root.val = findMin(root.right);
            root.right = deleteNode(root.right, root.val);
        }

        return root;
    }

    private int findMin(TreeNode node) {
```

Saved | Ln 1, Col 1

Testcase | Test Result

Graph - DFS

LeetCode 75

Description | Editorial | Solutions | Submissions

841. Keys and Rooms

Solved

Medium | Topics | Companies

There are `n` rooms labeled from `0` to `n - 1` and all the rooms are locked except for room `0`. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key.

When you visit a room, you may find a set of **distinct keys** in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms.

Given an array `rooms` where `rooms[i]` is the set of keys that you can obtain if you visited room `i`, return `true` *if you can visit **all** the rooms, or* `false` *otherwise.*

Example 1:

```
Input: rooms = [[1],[2],[3],[]]
Output: true
Explanation:
We visit room 0 and pick up key 1.
We then visit room 1 and pick up key 2.
We then visit room 2 and pick up key 3.
We then visit room 3.
Since we were able to visit every room, we return true.
```

Example 2:

```
Input: rooms = [[1,3],[3,0,1],[2],[0]]
Output: false
Explanation: We can not enter room number 2 since the only key that unlocks
it is in that room.
```

Constraints:

- `n == rooms.length`
- `2 <= n <= 1000`
- `0 <= rooms[i].length <= 1000`
- `1 <= sum(rooms[i].length) <= 3000`

6.6K | 145 | 24 Online

https://leetcode.com/problems/number-of-provinces?envType=study-plan-v2&e...

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.List;

class Solution {
    public boolean canVisitAllRooms(List<List<Integer>> rooms) {
        int n = rooms.size();
        boolean[] seen = new boolean[n];

        dfs(rooms, 0, seen);

        for (boolean visited : seen) {
            if (!visited) {
                return false;
            }
        }
        return true;
    }

    private void dfs(List<List<Integer>> rooms, int node, boolean[] seen) {
        seen[node] = true;
        for (int child : rooms.get(node)) {
            if (!seen[child]) {
                dfs(rooms, child, seen);
            }
        }
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

547. Number of Provinces

Solved

Medium | Topics | Companies

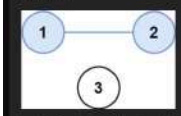
There are `n` cities. Some of them are connected, while some are not. If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the ith city and the jth city are directly connected, and `isConnected[i][j] = 0` otherwise.

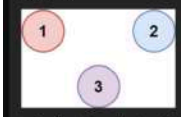
Return *the total number of* ***provinces***.

Example 1:




```
Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2
```

Example 2:




```
Input: isConnected = [[1,0,0],[0,1,0],[0,0,1]]
Output: 3
```

Constraints:

- `1 <= n <= 200`
- `n == isConnected.length`
- `n == isConnected[i].length`

10.7K | 184 | 128 Online

Submit | 0 | Premium

Code

Java | Auto

```java
class UnionFind {
    private int[] parent;

    public UnionFind(int totalNodes) {
        parent = new int[totalNodes];
        for (int node = 0; node < totalNodes; ++node) {
            parent[node] = node;
        }
    }

    public boolean union(int nodeA, int nodeB) {
        final int rootA = find(nodeA);
        final int rootB = find(nodeB);
        if (rootA == rootB) {
            return false; // No merge happened
        }
        parent[rootA] = rootB;
        return true; // Merge happened
    }

    private int find(int node) {
        if (parent[node] != node) {
            parent[node] = find(parent[node]); // Path compression
        }
        return parent[node];
    }
}

class Solution {
    public int findCircleNum(int[][] isConnected) {
        final int totalCities = isConnected.length;
        UnionFind unionFind = new UnionFind(totalCities);

        int componentCount = totalCities; // Start with all cities as separate components

        for (int cityA = 0; cityA < totalCities; ++cityA) {
            for (int cityB = cityA; cityB < totalCities; ++cityB) {
                if (isConnected[cityA][cityB] == 1 && unionFind.union(cityA, cityB)) {
                    componentCount--; // Only decrement when a merge actually happens
                }
            }
        }

        return componentCount;
    }
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

1466. Reorder Routes to Make All Paths Lead to the City Zero

Solved

Medium | Topics | Companies | Hint

There are `n` cities numbered from `0` to `n - 1` and `n - 1` roads such that there is only one way to travel between two different cities (this network form a tree). Last year, The ministry of transport decided to orient the roads in one direction because they are too narrow.

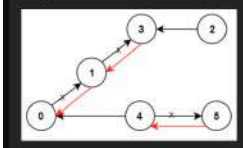
Roads are represented by `connections` where `connections[i] = [$a_i$, $b_i$]` represents a road from city a_i to city b_i.

This year, there will be a big event in the capital (city `0`), and many people want to travel to this city.

Your task consists of reorienting some roads such that each city can visit the city `0`. Return the **minimum** number of edges changed.

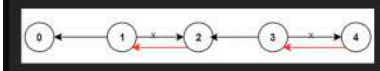
It's **guaranteed** that each city can reach city `0` after reorder.

Example 1:




```
Input: n = 6, connections = [[0,1],[1,3],[2,3],[4,0],[4,5]]
Output: 3
Explanation: Change the direction of edges show in red such that each node can reach
the node 0 (capital).
```

Example 2:




```
Input: n = 5, connections = [[1,0],[1,2],[3,2],[3,4]]
Output: 2
Explanation: Change the direction of edges show in red such that each node can reach
the node 0 (capital).
```

4.5K | 108 | 35 Online

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    // Adjacency list: each element = [neighbor, needsReversal]
    public int minReorder(int n, int[][] connections) {
        List<List<int[]>> graph = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            graph.add(new ArrayList<>());
        }

        for (int[] conn : connections) {
            int from = conn[0], to = conn[1];
            graph.get(from).add(new int[]{to, 1}); // needs reversal
            graph.get(to).add(new int[]{from, 0}); // already correct
        }

        boolean[] visited = new boolean[n];
        return dfs(0, visited, graph);
    }

    private int dfs(int node, boolean[] visited, List<List<int[]>> graph) {
        visited[node] = true;
        int reversals = 0;

        for (int[] edge : graph.get(node)) {
            int neighbor = edge[0];
            int needsReversal = edge[1];

            if (!visited[neighbor]) {
                reversals += needsReversal;
                reversals += dfs(neighbor, visited, graph);
            }
        }

        return reversals;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

399. Evaluate Division

Solved

Medium | Topics | Companies | Hint

You are given an array of variable pairs `equations` and an array of real numbers `values`, where `equations[i] = [$A_i$, $B_i$]` and `values[i]` represent the equation A_i / B_i = `values[i]`. Each A_i or B_i is a string that represents a single variable.

You are also given some `queries`, where `queries[j] = [$C_j$, $D_j$]` represents the jth query where you must find the answer for C_j / D_j = ?.

Return *the answers to all queries*. If a single answer cannot be determined, return `-1.0`.

Note: The input is always valid. You may assume that evaluating the queries will not result in division by zero and that there is no contradiction.

Note: The variables that do not occur in the list of equations are undefined, so the answer cannot be determined for them.

Example 1:

```
Input: equations = [["a","b"],["b","c"]], values = [2.0,3.0], queries = [["a","c"],
["b","a"],["a","e"],["a","a"],["x","x"]]
Output: [6.00000,0.50000,-1.00000,1.00000,-1.00000]
Explanation:
Given: a / b = 2.0, b / c = 3.0
queries are: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
return: [6.0, 0.5, -1.0, 1.0, -1.0 ]
note: x is undefined => -1.0
```

Example 2:

```
Input: equations = [["a","b"],["b","c"],["bc","cd"]], values = [1.5,2.5,5.0], queries =
[["a","c"],["c","b"],["bc","cd"],["cd","bc"]]
Output: [3.75000,0.40000,5.00000,0.20000]
```

Example 3:

```
Input: equations = [["a","b"]], values = [0.5], queries = [["a","b"],["b","a"],["a","c"],
["x","y"]]
Output: [0.50000,2.00000,-1.00000,-1.00000]
```

Constraints:

- `1 <= equations.length <= 20`
- `equations[i].length == 2`
- `1 <= $A_i$.length, $B_i$.length <= 5`
- `values.length == equations.length`
- `0.0 < values[i] <= 20.0`

9.9K | 256 | 199 Online

Submit | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public double[] calcEquation(List<List<String>> equations, double[] values, List<List<String>> queries) {
        // Step 1: Build the graph
        Map<String, Map<String, Double>> graph = new HashMap<>();

        for (int i = 0; i < equations.size(); i++) {
            String a = equations.get(i).get(0);
            String b = equations.get(i).get(1);
            double k = values[i];
            graph.putIfAbsent(a, new HashMap<>());
            graph.putIfAbsent(b, new HashMap<>());
            graph.get(a).put(b, k);
            graph.get(b).put(a, 1.0 / k);
        }
        // Step 2: Answer queries
        double[] results = new double[queries.size()];
        for (int i = 0; i < queries.size(); i++) {
            String src = queries.get(i).get(0);
            String dest = queries.get(i).get(1);
            if (!graph.containsKey(src) || !graph.containsKey(dest)) {
                results[i] = -1.0;
            } else if (src.equals(dest)) {
                results[i] = 1.0;
            } else {
                Set<String> visited = new HashSet<>();
                results[i] = dfs(graph, src, dest, 1.0, visited);
            }
        }
        return results;
    }

    private double dfs(Map<String, Map<String, Double>> graph, String current, String target, double accProduct, Set<String>
visited) {
        visited.add(current);
        if (current.equals(target)) {
            return accProduct;
        }
        for (Map.Entry<String, Double> neighbor : graph.get(current).entrySet()) {
            if (!visited.contains(neighbor.getKey())) {
                double result = dfs(graph, neighbor.getKey(), target, accProduct * neighbor.getValue(), visited);
                if (result != -1.0) {
                    return result;
                }
            }
        }

        return -1.0;
```

Saved | Ln 38, Col 10

Testcase | Test Result

Graph - BFS

LeetCode 75

Description | Editorial | Solutions | Submissions

1926. Nearest Exit from Entrance in Maze

Solved

Medium | Topics | Companies | Hint

You are given an `m x n` matrix `maze` (**0-indexed**) with empty cells (represented as `'.'`) and walls (represented as `'+'`). You also given the `entrance` of the maze, where `entrance = [entrance_row, entrance_col]` denotes the row and column of the cell you are initially standing at.

In one step, you can move one cell **up**, **down**, **left**, or **right**. You cannot step into a cell with a wall, and you cannot step outside the maze. Your goal is to find the **nearest exit** from the `entrance`. An **exit** is defined as an **empty cell** that is at the **border** of the `maze`. The `entrance` **does not count** as an exit.

Return *the **number of steps** in the shortest path from the* `entrance` *to the nearest exit, or* `-1` *if no such path exists.*

Example 1:

```
Input: maze = [["+","+",".","+","+"],[".",".",".","+",".],["+","+","+",".","."]], entrance = [1,2]
Output: 1
Explanation: There are 3 exits in this maze at [1,0], [0,2], and [2,3].
Initially, you are at the entrance cell [1,2].
- You can reach [1,0] by moving 2 steps left.
- You can reach [0,2] by moving 1 step up.
It is impossible to reach [2,3] from the entrance.
Thus, the nearest exit is [0,2], which is 1 step away.
```

Example 2:

2.5K | 85 | 203 Online

Submit | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public int nearestExit(char[][] maze, int[] entrance) {
        int rows = maze.length;
        int cols = maze[0].length;
        int entranceRow = entrance[0];
        int entranceCol = entrance[1];

        // Preferred order: right, down, left, up
        int[][] directions = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };

        Queue<int[]> queue = new ArrayDeque<>();
        queue.offer(new int[] { entranceRow, entranceCol, 0 }); // row, col, distance
        maze[entranceRow][entranceCol] = '+'; // mark entrance as visited

        while (!queue.isEmpty()) {
            int[] currentCell = queue.poll();
            int currentRow = currentCell[0];
            int currentCol = currentCell[1];
            int currentDistance = currentCell[2];

            // If it's a border cell (not the entrance), it's the exit
            if (!(currentRow == entranceRow && currentCol == entranceCol) &&
                    (currentRow == 0 || currentCol == 0 ||
                            currentRow == rows - 1 || currentCol == cols - 1)) {
                return currentDistance;
            }

            // Explore in preferred direction order
            for (int[] direction : directions) {
                int nextRow = currentRow + direction[0];
                int nextCol = currentCol + direction[1];

                if (nextRow >= 0 && nextRow < rows &&
                        nextCol >= 0 && nextCol < cols &&
                        maze[nextRow][nextCol] == '.') {

                    maze[nextRow][nextCol] = '+'; // mark visited
                    queue.offer(new int[] { nextRow, nextCol, currentDistance + 1 });
                }
            }
        }

        return -1; // no exit found
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

994. Rotting Oranges

Solved

Medium | Topics | Companies

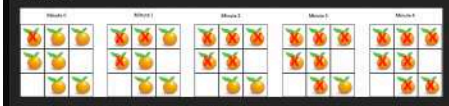
You are given an `m x n` `grid` where each cell can have one of three values:

- `0` representing an empty cell,
- `1` representing a fresh orange, or
- `2` representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this is impossible, return `-1`.

Example 1:



```
Input: grid = [[2,1,1],[1,1,0],[0,1,1]]
Output: 4
```

Example 2:

```
Input: grid = [[2,1,1],[0,1,1],[1,0,1]]
Output: -1
Explanation: The orange in the botton left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.
```

Example 3:

```
Input: grid = [[0,2]]
Output: 0
Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 10`
- `grid[i][j]` is `0`, `1`, or `2`.

14.3K | 215 | 206 Online

Submit | Premium

Code

Java | Auto

```java
import java.util.LinkedList;
import java.util.Queue;

public class Solution {

    private static final int[][] DIRECTIONS = {
            {0, 1}, {1, 0}, {0, -1}, {-1, 0}
    };

    public static int orangesRotting(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        Queue<int[]> queue = new LinkedList<>();
        int freshOranges = 0;

        // Step 1: Initialize the queue with all initially rotten oranges
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == 2) {
                    queue.offer(new int[]{i, j});
                } else if (grid[i][j] == 1) {
                    freshOranges++;
                }
            }
        }

        if (freshOranges == 0) return 0;

        int minutes = -1;

        // Step 2: BFS traversal
        while (!queue.isEmpty()) {
            int size = queue.size();
            minutes++;
            for (int k = 0; k < size; k++) {
                int[] cell = queue.poll();
                int row = cell[0];
                int col = cell[1];

                for (int[] dir : DIRECTIONS) {
                    int newRow = row + dir[0];
                    int newCol = col + dir[1];

                    if (isValid(newRow, newCol, m, n, grid)) {
                        grid[newRow][newCol] = 2;
                        freshOranges--;
                        queue.offer(new int[]{newRow, newCol});
                    }
                }
```

Saved | Ln 1, Col 1

Testcase | Test Result

Heap / Priority Queue

LeetCode 75

Description | Editorial | Solutions | Submissions

215. Kth Largest Element in an Array

Solved

Medium | Topics | Companies

Given an integer array `nums` and an integer `k`, return the kth *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2:

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Constraints:

- `1 <= k <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`

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

Yes | No

Accepted 3,155,346/4.6M | Acceptance Rate 68.3%

Topics

Companies

Similar Questions

Discussion (337)

18.1K | 337 | 204 Online

Submit | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public int findKthLargest(int[] nums, int k) {
        PriorityQueue<Integer> minHeap = new PriorityQueue<>(); // min-heap by default in Java

        for (int num : nums) {
            minHeap.offer(num);
            if (minHeap.size() > k) {
                minHeap.poll(); // remove smallest to keep only k largest elements
            }
        }

        return minHeap.peek(); // top of heap = kth largest
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

2336. Smallest Number in Infinite Set

Solved

Medium | Topics | Companies | Hint

You have a set which contains all positive integers `[1, 2, 3, 4, 5, ...]`.

Implement the `SmallestInfiniteSet` class:

- `SmallestInfiniteSet()` initializes the **SmallestInfiniteSet** object to contain **all** positive integers.
- `int popSmallest()` **Removes** and returns the smallest integer contained in the infinite set.
- `void addBack(int num)` **Adds** a positive integer `num` back into the infinite set, if it is **not** already in the infinite set.

Example 1:

```
Input
["SmallestInfiniteSet", "addBack", "popSmallest", "popSmallest", "popSmallest", "addBack",
"popSmallest", "popSmallest", "popSmallest"]
[[], [2], [], [], [], [3], [], [], []]
Output
[null, null, 1, 2, 3, null, 1, 4, 5]

Explanation
SmallestInfiniteSet smallestInfiniteSet = new SmallestInfiniteSet();
smallestInfiniteSet.addBack(2);    // 2 is already in the set, so no change is made.
smallestInfiniteSet.popSmallest(); // return 1, since 1 is the smallest number, and remove
it from the set.
smallestInfiniteSet.popSmallest(); // return 2, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 3, and remove it from the set.
smallestInfiniteSet.addBack(1);    // 1 is added back to the set.
smallestInfiniteSet.popSmallest(); // return 1, since 1 was added back to the set and
                                   // is the smallest number, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 4, and remove it from the set.
smallestInfiniteSet.popSmallest(); // return 5, and remove it from the set.
```

Constraints:

- `1 <= num <= 1000`
- At most `1000` calls will be made **in total** to `popSmallest` and `addBack`.

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

Yes | No

1.8K | 75 | 13 Online

Submit | Premium

Code

Java | Auto

```java
import java.util.*;

class SmallestInfiniteSet {
    private int nextSmallest;
    private PriorityQueue<Integer> pq;
    private Set<Integer> inPQ;

    public SmallestInfiniteSet() {
        nextSmallest = 1; // Initially smallest number is 1
        pq = new PriorityQueue<>();
        inPQ = new HashSet<>();
    }

    public int popSmallest() {
        if (!pq.isEmpty()) {
            int smallest = pq.poll();
            inPQ.remove(smallest);
            return smallest;
        }
        return nextSmallest++;
    }

    public void addBack(int num) {
        // Add only if the number is less than nextSmallest
        // (already popped) and not already in pq
        if (num < nextSmallest && !inPQ.contains(num)) {
            pq.offer(num);
            inPQ.add(num);
        }
    }
}

/**
 * Your SmallestInfiniteSet object will be instantiated and called as such:
 * SmallestInfiniteSet obj = new SmallestInfiniteSet();
 * int param_1 = obj.popSmallest();
 * obj.addBack(num);
 */
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

2542. Maximum Subsequence Score

Solved

Medium | Topics | Companies | Hint

You are given two **0-indexed** integer arrays `nums1` and `nums2` of equal length `n` and a positive integer `k`. You must choose a **subsequence** of indices from `nums1` of length `k`.

For chosen indices $i_0, i_1, ..., i_{k-1}$, your **score** is defined as:

- The sum of the selected elements from `nums1` multiplied with the **minimum** of the selected elements from `nums2`.
- It can defined simply as: `(nums1[i0] + nums1[i1] +...+ nums1[ik-1]) * min(nums2[i0] , nums2[i1], ... ,nums2[ik-1])`.

Return *the **maximum** possible score.*

A **subsequence** of indices of an array is a set that can be derived from the set `{0, 1, ..., n-1}` by deleting some or no elements.

Example 1:

```
Input: nums1 = [1,3,3,2], nums2 = [2,1,3,4], k = 3
Output: 12
Explanation:
The four possible subsequence scores are:
- We choose the indices 0, 1, and 2 with score = (1+3+3) * min(2,1,3) = 7.
- We choose the indices 0, 1, and 3 with score = (1+3+2) * min(2,1,4) = 6.
- We choose the indices 0, 2, and 3 with score = (1+3+2) * min(2,3,4) = 12.
- We choose the indices 1, 2, and 3 with score = (3+3+2) * min(1,3,4) = 8.
Therefore, we return the max score, which is 12.
```

Example 2:

```
Input: nums1 = [4,2,3,1,1], nums2 = [7,5,10,9,6], k = 1
Output: 30
Explanation:
Choosing index 2 is optimal: nums1[2] * nums2[2] = 3 * 10 = 30 is the maximum possible
score.
```

Constraints:

- `n == nums1.length == nums2.length`
- `1 <= n <= 10^5`
- `0 <= nums1[i], nums2[i] <= 10^5`
- `1 <= k <= n`

2K | 119 | 200 Online

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public long maxScore(int[] nums1, int[] nums2, int k) {
        int n = nums1.length;

        int[][] pairs = new int[n][2];
        for (int i = 0; i < n; i++) {
            pairs[i][0] = nums2[i];
            pairs[i][1] = nums1[i];
        }

        Arrays.sort(pairs, (a, b) -> b[0] - a[0]);

        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
        long sum = 0, maxScore = 0;

        for (int[] pair : pairs) {
            int n2 = pair[0];
            int n1 = pair[1];

            // Add current nums1 to heap and sum
            minHeap.offer(n1);
            sum += n1;

            // Keep only top k nums1
            if (minHeap.size() > k) {
                sum -= minHeap.poll();
            }

            // If exactly k elements chosen, calculate score
            if (minHeap.size() == k) {
                maxScore = Math.max(maxScore, sum * n2);
            }
        }

        return maxScore;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

2462. Total Cost to Hire K Workers

Solved

Medium | Topics | Companies | Hint

You are given a **0-indexed** integer array `costs` where `costs[i]` is the cost of hiring the ith worker.

You also also given two integers `k` and `candidates`. We want to hire exactly `k` workers according to the following rules:

- You will run `k` sessions and hire exactly one worker in each session.
- In each hiring session, choose the worker with the lowest cost from either the first `candidates` workers or the last `candidates` workers. Break the tie by the smallest index.
  - For example, if `costs = [3,2,7,7,1,2]` and `candidates = 2`, then in the first hiring session, we will choose the 4th worker because they have the lowest cost `[3,2,7,7,1,2]`.
  - In the second hiring session, we will choose 1st worker because they have the same lowest cost as 4th worker but they have the smallest index `[3,2,7,7,2]`. Please note that the indexing may be changed in the process.
- If there are fewer than candidates workers remaining, choose the worker with the lowest cost among them. Break the tie by the smallest index.
- A worker can only be chosen once.

Return *the total cost to hire exactly `k` workers.*

Example 1:

```
Input: costs = [17,12,10,2,7,2,11,20,8], k = 3, candidates = 4
Output: 11
Explanation: We hire 3 workers in total. The total cost is initially 0.
- In the first hiring round we choose the worker from [17,12,10,2,7,2,11,20,8]. The lowest
cost is 2, and we break the tie by the smallest index, which is 3. The total cost = 0 + 2 =
2.
- In the second hiring round we choose the worker from [17,12,10,7,2,11,20,8]. The lowest
cost is 2 (index 4). The total cost = 2 + 2 = 4.
- In the third hiring round we choose the worker from [17,12,10,7,11,20,8]. The lowest cost
is 7 (index 3). The total cost = 4 + 7 = 11. Notice that the worker with index 3 was common
in the first and last four workers.
The total hiring cost is 11.
```

Example 2:

```
Input: costs = [1,2,4,1], k = 3, candidates = 3
Output: 4
Explanation: We hire 3 workers in total. The total cost is initially 0.
- In the first hiring round we choose the worker from [1,2,4,1]. The lowest cost is 1, and
we break the tie by the smallest index, which is 0. The total cost = 0 + 1 = 1. Notice that
workers with index 1 and 2 are common in the first and last 3 workers.
- In the second hiring round we choose the worker from [2,4,1]. The lowest cost is 1 (index
2). The total cost = 1 + 1 = 2.
- In the third hiring round there are less than three candidates. We choose the worker from
the remaining workers [2,4]. The lowest cost is 2 (index 0). The total cost = 2 + 2 = 4.
```

2K | 186 | 199 Online

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.PriorityQueue;

class Solution {
    public long totalCost(int[] costs, int k, int candidates) {
        int n = costs.length;
        int left = 0;
        int right = n - 1;
        long total = 0;

        PriorityQueue<Integer> pqLeft = new PriorityQueue<>();
        PriorityQueue<Integer> pqRight = new PriorityQueue<>();

        for (int i = 0; i < candidates && left <= right; i++) {
            pqLeft.offer(costs[left++]);
        }
        for (int i = 0; i < candidates && left <= right; i++) {
            pqRight.offer(costs[right--]);
        }

        for (int hire = 0; hire < k; hire++) {
            int leftCost = pqLeft.isEmpty() ? Integer.MAX_VALUE : pqLeft.peek();
            int rightCost = pqRight.isEmpty() ? Integer.MAX_VALUE : pqRight.peek();

            if (leftCost <= rightCost) {
                total += pqLeft.poll();
                if (left <= right) {
                    pqLeft.offer(costs[left++]);
                }
            } else {
                total += pqRight.poll();
                if (left <= right) {
                    pqRight.offer(costs[right--]);
                }
            }
        }

        return total;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

Binary Search

LeetCode 75

Description | Editorial | Solutions | Submissions

374. Guess Number Higher or Lower

Solved

Easy | Topics | Companies

We are playing the Guess Game. The game is as follows:

I pick a number from `1` to `n`. You have to guess which number I picked.

Every time you guess wrong, I will tell you whether the number I picked is higher or lower than your guess.

You call a pre-defined API `int guess(int num)`, which returns three possible results:

- `-1`: Your guess is higher than the number I picked (i.e. `num > pick`).
- `1`: Your guess is lower than the number I picked (i.e. `num < pick`).
- `0`: your guess is equal to the number I picked (i.e. `num == pick`).

Return *the number that I picked.*

Example 1:

```
Input: n = 10, pick = 6
Output: 6
```

Example 2:

```
Input: n = 1, pick = 1
Output: 1
```

Example 3:

```
Input: n = 2, pick = 1
Output: 1
```

Constraints:

- $1 \le n \le 2^{31} - 1$
- $1 \le pick \le n$

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

Yes | No

Accepted 889,284/1.6M Acceptance Rate 56.1%

4.1K | 230 | 33 Online

Submit | 0 | Premium

Code

Java | Auto

```java
/**
 * Forward declaration of guess API.
 * @param  num   your guess
 * @return      -1 if num is higher than the picked number
 *               1 if num is lower than the picked number
 *              otherwise return 0
 * int guess(int num);
 */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        int left = 1, right = n;

        while (left <= right) {
            int mid = left + (right - left) / 2; // Prevents overflow

            int result = guess(mid);
            if (result == 0) {
                return mid;  // Found the correct number
            } else if (result == -1) {
                right = mid - 1;  // Target is smaller
            } else {
                left = mid + 1;  // Target is larger
            }
        }

        return -1; // This line should never be reached
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

2300. Successful Pairs of Spells and Potions

Solved

Medium | Topics | Companies | Hint

You are given two positive integer arrays `spells` and `potions`, of length `n` and `m` respectively, where `spells[i]` represents the strength of the ith spell and `potions[j]` represents the strength of the jth potion.

You are also given an integer `success`. A spell and potion pair is considered **successful** if the **product** of their strengths is **at least** `success`.

*Return an integer array `pairs` of length `n` where `pairs[i]` is the number of **potions** that will form a successful pair with the ith spell.*

Example 1:

```
Input: spells = [5,1,3], potions = [1,2,3,4,5], success = 7
Output: [4,0,3]
Explanation:
- 0th spell: 5 * [1,2,3,4,5] = [5,10,15,20,25]. 4 pairs are successful.
- 1st spell: 1 * [1,2,3,4,5] = [1,2,3,4,5]. 0 pairs are successful.
- 2nd spell: 3 * [1,2,3,4,5] = [3,6,9,12,15]. 3 pairs are successful.
Thus, [4,0,3] is returned.
```

Example 2:

```
Input: spells = [3,1,2], potions = [8,5,8], success = 16
Output: [2,0,2]
Explanation:
- 0th spell: 3 * [8,5,8] = [24,15,24]. 2 pairs are successful.
- 1st spell: 1 * [8,5,8] = [8,5,8]. 0 pairs are successful.
- 2nd spell: 2 * [8,5,8] = [16,10,16]. 2 pairs are successful.
Thus, [2,0,2] is returned.
```

Constraints:

- `n == spells.length`
- `m == potions.length`
- `1 <= n, m <= 10^5`
- `1 <= spells[i], potions[i] <= 10^5`
- `1 <= success <= 10^10`

https://leetcode.com/problems/find-peak-element?envType=study-plan-v2&envId=...

201 Online

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.Arrays;

class Solution {
    public int[] successfulPairs(int[] spells, int[] potions, long success) {
        Arrays.sort(potions);
        int n = spells.length;
        int m = potions.length;
        int[] result = new int[n];

        for (int i = 0; i < n; i++) {
            long minPotion = (success + spells[i] - 1) / spells[i];
            int idx = binarySearch(potions, minPotion);
            result[i] = m - idx;
        }
        return result;
    }

    private int binarySearch(int[] arr, long target) {
        int left = 0, right = arr.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

162. Find Peak Element

Solved

Medium | Topics | Companies

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or
index number 5 where the peak element is 6.
```

Constraints:

- `1 <= nums.length <= 1000`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `nums[i] != nums[i + 1]` for all valid `i`.

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

Yes | No

Accepted 2,053,586/4.4M Acceptance Rate 46.5%

Topics

Companies

13.7K | 393

170 Online

Submit | 0 | Premium

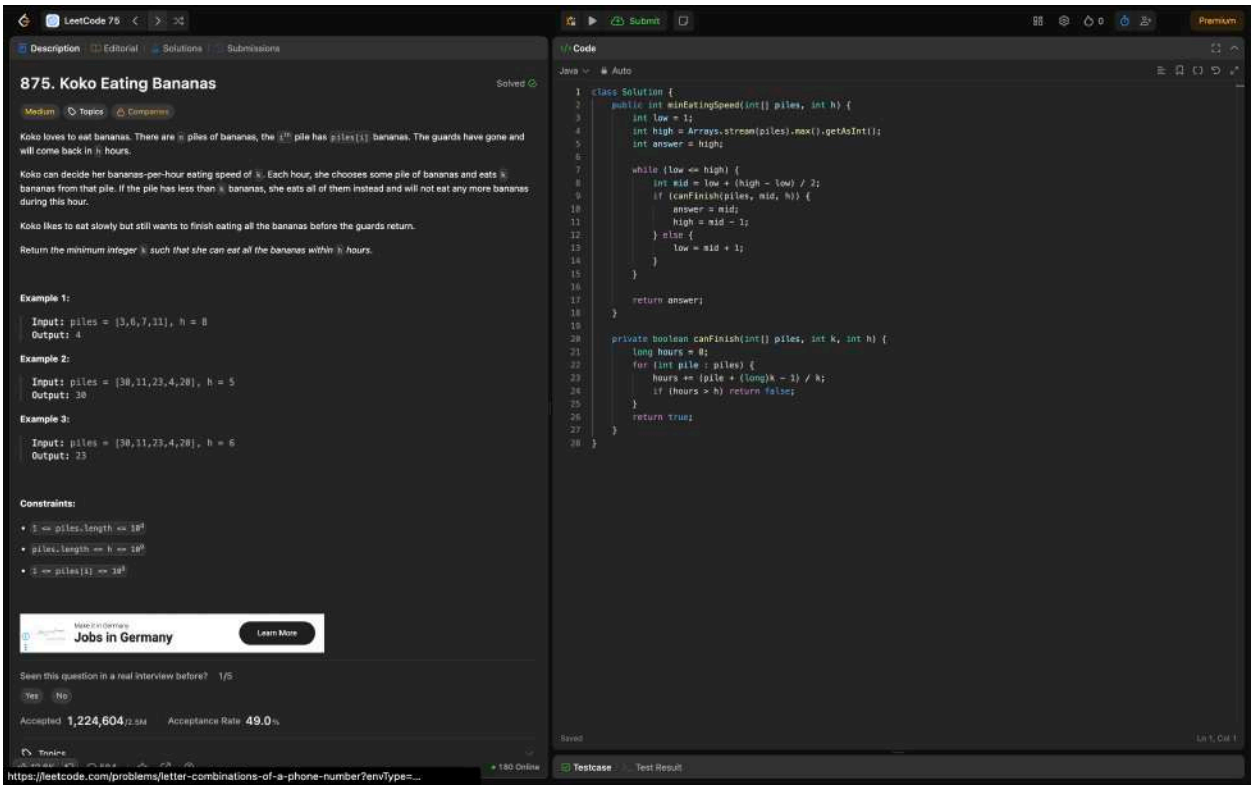
Code

Java | Auto

```java
class Solution {
    public int findPeakElement(int[] nums) {
        int left = 0, right = nums.length - 1;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] < nums[mid + 1]) {
                // Peak lies in the right half
                left = mid + 1;
            } else {
                // Peak lies in the left half (including mid)
                right = mid;
            }
        }

        // left == right == peak index
        return left;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result



Backtracking

LeetCode 75

Description | Editorial | Solutions | Submissions

17. Letter Combinations of a Phone Number

Solved

Medium | Topics | Companies

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

Example 3:

```
Input: digits = "2"
Output: ["a","b","c"]
```

Constraints:

- `0 <= digits.length <= 4`
- `digits[i]` is a digit in the range `['2', '9']`.

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

20.9K | 311 | 221 Online

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    private static final String[] KEYPAD = {
        "",     // 0
        "",     // 1
        "abc",  // 2
        "def",  // 3
        "ghi",  // 4
        "jkl",  // 5
        "mno",  // 6
        "pqrs", // 7
        "tuv",  // 8
        "wxyz"  // 9
    };

    public List<String> letterCombinations(String digits) {
        List<String> result = new ArrayList<>();
        if (digits == null || digits.length() == 0) return result;

        backtrack(digits, 0, new StringBuilder(), result);
        return result;
    }

    private void backtrack(String digits, int index, StringBuilder current, List<String> result) {
        if (index == digits.length()) {
            result.add(current.toString());
            return;
        }

        String letters = KEYPAD[digits.charAt(index) - '0'];
        for (char ch : letters.toCharArray()) {
            current.append(ch);
            backtrack(digits, index + 1, current, result);
            current.deleteCharAt(current.length() - 1); // undo choice
        }
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

216. Combination Sum III

Solved

Medium | Topics | Companies

Find all valid combinations of `k` numbers that sum up to `n` such that the following conditions are true:

- Only numbers `1` through `9` are used.
- Each number is used **at most once**.

Return *a list of all possible valid combinations*. The list must not contain the same combination twice, and the combinations may be returned in any order.

Example 1:

```
Input: k = 3, n = 7
Output: [[1,2,4]]
Explanation:
1 + 2 + 4 = 7
There are no other valid combinations.
```

Example 2:

```
Input: k = 3, n = 9
Output: [[1,2,6],[1,3,5],[2,3,4]]
Explanation:
1 + 2 + 6 = 9
1 + 3 + 5 = 9
2 + 3 + 4 = 9
There are no other valid combinations.
```

Example 3:

```
Input: k = 4, n = 1
Output: []
Explanation: There are no valid combinations.
Using 4 different numbers in the range [1,9], the smallest sum we can get is 1+2+3+4 = 10
and since 10 > 1, there are no valid combination.
```

Constraints:

- `2 <= k <= 9`
- `1 <= n <= 60`

6.4K | 100 | 51 Online

Submit | 0 | Premium

Code

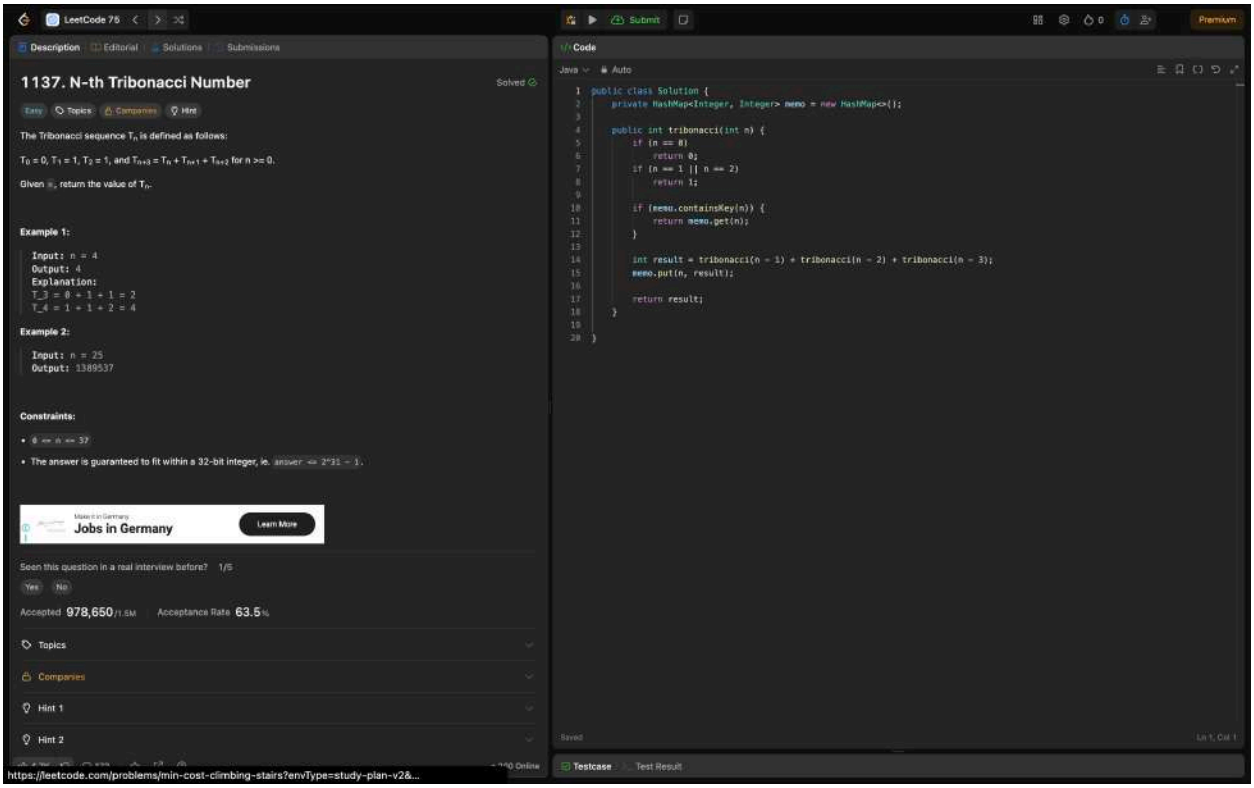
Java | Auto

```java
import java.util.*;

class Solution {
    public List<List<Integer>> combinationSum3(int k, int n) {
        List<List<Integer>> result = new ArrayList<>();
        backtrack(1, k, n, new ArrayList<>(), result);
        return result;
    }

    private void backtrack(int start, int k, int remainingSum, List<Integer> current, List<List<Integer>> result) {
        // If we've chosen k numbers
        if (current.size() == k) {
            if (remainingSum == 0) {
                result.add(new ArrayList<>(current));
            }
            return;
        }

        // Try each number from 'start' to 9
        for (int num = start; num <= 9; num++) {
            if (num > remainingSum) break; // pruning

            current.add(num); // choose
            backtrack(num + 1, k, remainingSum - num, current, result);
            current.remove(current.size() - 1); // undo choice
        }
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

DP - 1D


LeetCode 75
Description
Editorial
Solutions
Submissions
1137. N-th Tribonacci Number
Solved
Easy
Topics
Companies
Hint
The Tribonacci sequence T_n is defined as follows:
$T_0 = 0$, $T_1 = 1$, $T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for $n >= 0$.
Given n, return the value of T_n.
Example 1:
Input: n = 4
Output: 4
Explanation:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4
Example 2:
Input: n = 25
Output: 1389537
Constraints:
0 <= n <= 37
The answer is guaranteed to fit within a 32-bit integer, ie. answer <= 2^31 - 1.
Jobs in Germany
Learn More
Seen this question in a real interview before? 1/5
Yes
No
Accepted 978,650/1.5M
Acceptance Rate 63.5%
Topics
Companies
Hint 1
Hint 2
https://leetcode.com/problems/min-cost-climbing-stairs?envType=study-plan-v2&...
Submit
Premium
Code
Java
Auto
public class Solution {
private HashMap<Integer, Integer> memo = new HashMap<>();
public int tribonacci(int n) {
if (n == 0)
return 0;
if (n == 1 || n == 2)
return 1;
if (memo.containsKey(n)) {
return memo.get(n);
}
int result = tribonacci(n - 1) + tribonacci(n - 2) + tribonacci(n - 3);
memo.put(n, result);
return result;
}
}
Saved
Ln 1, Col 1
Testcase
Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

746. Min Cost Climbing Stairs

Solved

Easy | Topics | Companies | Hint

You are given an integer array `cost` where `cost[i]` is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor.*

Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```

Example 2:

```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```

Constraints:

- `2 <= cost.length <= 1000`
- `0 <= cost[i] <= 999`

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

Yes | No

Accepted 1,549,803/2.3M Acceptance Rate 67.4%

95 Online

https://leetcode.com/problems/house-robber?envType=study-plan-v2&envId=leetcode-75

Submit | Premium

Code

Java | Auto

```java
class Solution {
    public int minCostClimbingStairs(int[] cost) {
        int n = cost.length;

        // dp[i] will represent the minimum cost to reach step i
        int[] dp = new int[n + 1];

        // Base cases: we can start from step 0 or 1
        dp[0] = 0;
        dp[1] = 0;

        for (int i = 2; i <= n; i++) {
            dp[i] = Math.min(
                dp[i - 1] + cost[i - 1],
                dp[i - 2] + cost[i - 2]
            );
        }

        return dp[n];
    }
}
```

Saved Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

198. House Robber

Solved

Medium | Topics | Companies

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night.**

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight **without alerting the police**.*

Example 1:

```
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
```

Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 400`

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

Yes | No

Accepted 3,063,170/5.8M Acceptance Rate 52.5%

Topics

Companies

Similar Questions

270 Online

https://leetcode.com/problems/domino-and-tromino-tiling?envType=study-plan-...

Submit | Premium

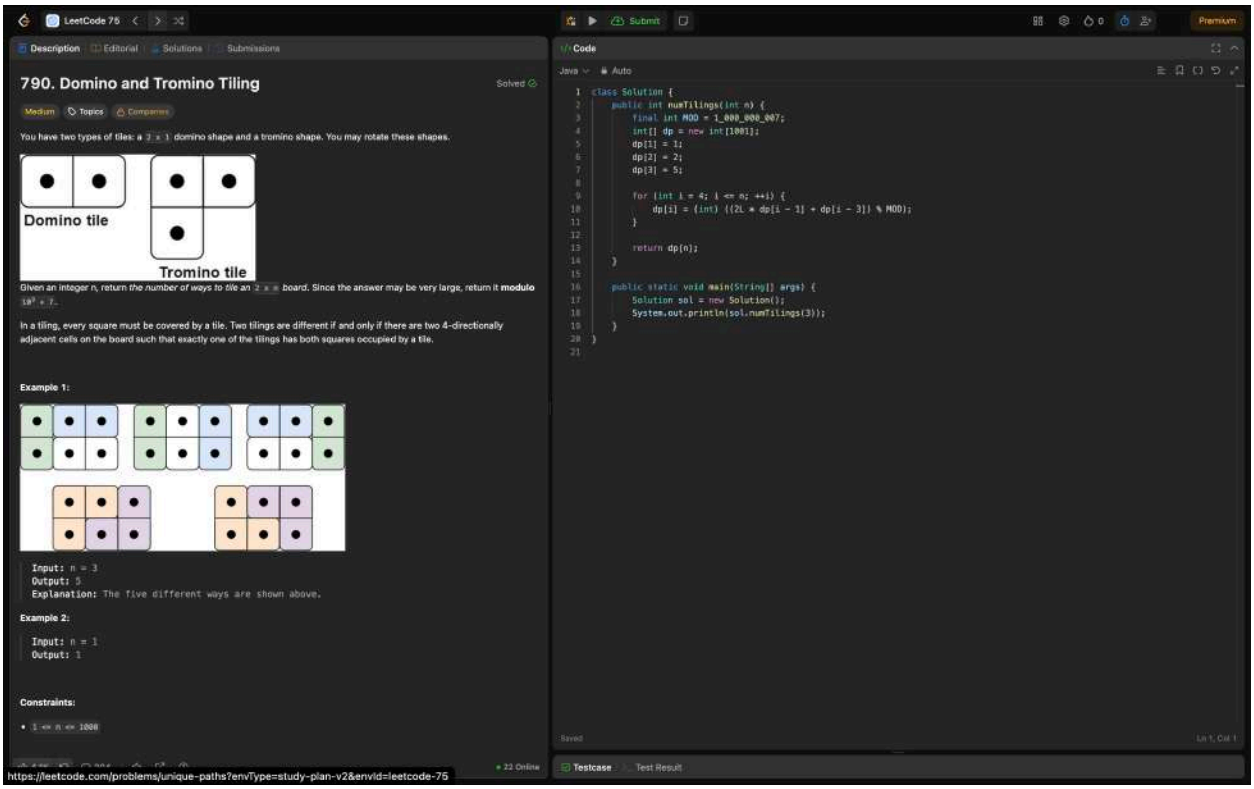
Code

Java | Auto

```java
class Solution {
    // dp[i] represents the maximum amount of money that can be robbed from the
    // first i houses.
    public int rob(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        int[] dp = new int[nums.length + 1];
        dp[0] = 0;
        dp[1] = nums[0];
        for (int i = 2; i <= nums.length; i++) {
            dp[i] = Math.max(nums[i - 1] + dp[i - 2], dp[i - 1]);
        }
        System.out.println(Arrays.toString(dp));
        return dp[nums.length];
    }
}
```

Saved Ln 1, Col 1

Testcase | Test Result



DP - MultiDimensional

LeetCode 75

Description | Editorial | Solutions | Submissions

62. Unique Paths

Solved

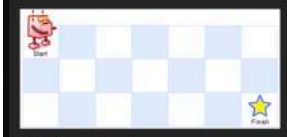
Medium | Topics | Companies

There is a robot on an `m x n` grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, *return the number of possible unique paths that the robot can take to reach the bottom-right corner.*

The test cases are generated so that the answer will be less than or equal to `2 * 10^9`.

Example 1:




```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Constraints:

- `1 <= m, n <= 100`

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

17.8K | 213 | 194 Online

Submit | Premium

Code

Java | Auto

```java
class Solution {
    int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
        for (int[] dp1 : dp)
            Arrays.fill(dp1, 1);
        for (int i = 1; i < m; i++)
            for (int j = 1; j < n; j++)
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
        return dp[m - 1][n - 1];
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

1143. Longest Common Subsequence

Solved

Medium | Topics | Companies | Hint

Given two strings `text1` and `text2`, return *the length of their longest* **common subsequence**. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

Constraints:

- `1 <= text1.length, text2.length <= 1000`
- `text1` and `text2` consist of only lowercase English characters.

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

Yes | No

Accepted **1,570,611** /2.7M Acceptance Rate **58.4**%

192 Online

leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee?e...

Submit | Premium

Code

Java | Auto

```java
class Solution {
    public int longestCommonSubsequence(String text1, String text2) {
        int m = text1.length();
        int n = text2.length();
        // dp[i][j] = length of LCS(text1[0..i], text2[0..j])
        int[][] dp = new int[m + 1][n + 1];

        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if (text1.charAt(i) == text2.charAt(j)) {
                    dp[i + 1][j + 1] = 1 + dp[i][j];
                } else {
                    dp[i + 1][j + 1] = Math.max(dp[i][j + 1], dp[i + 1][j]);
                }
            }
        }

        return dp[m][n];
    }

    public static void main(String[] args) {
        Solution sol = new Solution();
        String text1 = "abcde";
        String text2 = "ace";
        System.out.println(sol.longestCommonSubsequence(text1, text2)); // Output: 3
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

714. Best Time to Buy and Sell Stock with Transaction Fee

Solved

Medium | Topics | Companies | Hint

You are given an array `prices` where `prices[i]` is the price of a given stock on the ith day, and an integer `fee` representing a transaction fee.

Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction.

Note:

- You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).
- The transaction fee is only charged once for each stock purchase and sale.

Example 1:

```
Input: prices = [1,3,2,8,4,9], fee = 2
Output: 8
Explanation: The maximum profit can be achieved by:
- Buying at prices[0] = 1
- Selling at prices[3] = 8
- Buying at prices[4] = 4
- Selling at prices[5] = 9
The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.
```

Example 2:

```
Input: prices = [1,3,7,5,10,3], fee = 3
Output: 6
```

Constraints:

- $1 \le$ `prices.length` $\le 5 \times 10^4$
- $1 \le$ `prices[i]` $\le 5 \times 10^4$
- $0 \le$ `fee` $< 5 \times 10^4$

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

Yes | No

Accepted 507,947/716.1K | Acceptance Rate 70.9%

39 Online

https://leetcode.com/problems/edit-distance?envType=study-plan-v2&envId=lee...

Submit | 0 | Premium

Code

Java | Auto

```java
class Solution {
    public int maxProfit(int[] prices, int fee) {
        int n = prices.length;
        if (n == 0) return 0;

        // dp[i][0] = max profit on day i when we do NOT hold a stock
        // dp[i][1] = max profit on day i when we DO hold a stock
        int[][] dp = new int[n][2];

        dp[0][0] = 0;           // No stock on first day
        dp[0][1] = -prices[0];  // Buy stock on first day

        for (int i = 1; i < n; i++) {
            // Either we do nothing (dp[i-1][0]) or sell stock today
            dp[i][0] = Math.max(dp[i - 1][0], dp[i - 1][1] + prices[i] - fee);
            // Either we do nothing (dp[i-1][1]) or buy stock today
            dp[i][1] = Math.max(dp[i - 1][1], dp[i - 1][0] - prices[i]);
        }

        return dp[n - 1][0]; // Profit is always higher when not holding stock at the end
    }

    public static void main(String[] args) {
        Solution sol = new Solution();
        int[] prices = {1, 3, 2, 8, 4, 9};
        int fee = 2;
        System.out.println(sol.maxProfit(prices, fee)); // Output: 8
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

72. Edit Distance

Solved

Medium | Topics | Companies

Given two strings `word1` and `word2`, *return the minimum number of operations required to convert* `word1` *to* `word2`.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

Constraints:

- $0 \le$ `word1.length, word2.length` ≤ 500
- `word1` and `word2` consist of lowercase English letters.

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

Yes | No

Accepted 1,231,385/2.1M | Acceptance Rate 59.2%

186 Online

https://leetcode.com/problems/counting-bits?envType=study-plan-v2&envId=lee...

Submit | 0 | Premium

Code

Java | Auto

```java
class Solution {
    //  dp[i][j] represents the minimum number of operations (edits) required to transform the substring word1[0...i-1] to the substring word2[0...j-1].
    public int minDistance(String word1, String word2) {
        int m = word1.length();
        int n = word2.length();
        if (m == 0) return n;
        if (n == 0) return m;
        int[][] dp = new int[m + 1][n + 1];

        for (int i = 1; i <= m; i++) {
            dp[i][0] = i;
        }

        for (int j = 1; j <= n; j++) {
            dp[0][j] = j;
        }

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (word1.charAt(i - 1) == word2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1];
                } else {
                    dp[i][j] = Math.min(dp[i - 1][j - 1], // replace
                            Math.min(dp[i - 1][j], // remove
                                    dp[i][j - 1] // insert
                            )) + 1;
                }
            }
        }

        System.out.println(Arrays.deepToString(dp));
        return dp[m][n];
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

Bit Manipulation

LeetCode 75

Description | Editorial | Solutions | Submissions

338. Counting Bits

Solved

Easy | Topics | Companies | Hint

Given an integer `n`, return an array `ans` of length `n + 1` such that for each `i` (`0 <= i <= n`), `ans[i]` is the **number of** `1`'s in the binary representation of `i`.

Example 1:

```
Input: n = 2
Output: [0,1,1]
Explanation:
0 --> 0
1 --> 1
2 --> 10
```

Example 2:

```
Input: n = 5
Output: [0,1,1,2,1,2]
Explanation:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
```

Constraints:

- `0 <= n <= 10^5`

Follow up:

- It is very easy to come up with a solution with a runtime of `O(n log n)`. Can you do it in linear time `O(n)` and possibly in a single pass?
- Can you do it without using any built-in function (i.e., like `__builtin_popcount` in C++)?

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

Yes | No

190 Online

https://leetcode.com/problems/single-number?envType=study-plan-v2&envId=le...

Submit | Premium

Code

Java | Auto

```java
class Solution {
    public int[] countBits(int n) {
        int[] dp = new int[n + 1];
        dp[0] = 0;

        for (int i = 1; i <= n; i++) {
            // Number of 1's in i = Number of 1's in i/2 + last bit
            dp[i] = dp[i >> 1] + (i & 1);
        }

        return dp;
    }

    public static void main(String[] args) {
        Solution sol = new Solution();
        int[] result = sol.countBits(5);
        for (int num : result) {
            System.out.print(num + " ");
        }
        // Output: 0 1 1 2 1 2
    }
}
```

Saved

Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

136. Single Number

Solved

Easy | Topics | Companies | Hint

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

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

Example 2:

```
Input: nums = [4,1,2,1,2]
Output: 4
```

Example 3:

```
Input: nums = [1]
Output: 1
```

Constraints:

- `1 <= nums.length <= 3 * 10^4`
- `-3 * 10^4 <= nums[i] <= 3 * 10^4`
- Each element in the array appears twice except for one element which appears only once.

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

Yes | No

Accepted 3,909,268/5.1M | Acceptance Rate 76.4%

Topics

Companies

191 Online

https://leetcode.com/problems/minimum-flips-to-make-a-or-b-equal-to-c?envTy...

Submit | Premium

Code

Java | Auto

```java
class Solution {
    public int singleNumber(int[] nums) {
        int result = 0;

        // XOR all numbers
        for (int num : nums) {
            result ^= num;
        }

        return result;
    }

    public static void main(String[] args) {
        Solution sol = new Solution();
        int[] nums = {4, 1, 2, 1, 2};
        System.out.println(sol.singleNumber(nums)); // Output: 4
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

1318. Minimum Flips to Make a OR b Equal to c

Solved

Medium | Topics | Companies | Hint

Given 3 positives numbers `a`, `b` and `c`. Return the minimum flips required in some bits of `a` and `b` to make (`a` OR `b` == `c`). (bitwise OR operation).
Flip operation consists of change **any** single bit 1 to 0 or change the bit 0 to 1 in their binary representation.

Example 1:

```
0010 -> a          0001 -> a
0110 -> b    ->    0100 -> b
----               ----
0101 -> c          0101 -> c
```

```
Input: a = 2, b = 6, c = 5
Output: 3
Explanation: After flips a = 1 , b = 4 , c = 5 such that (a OR b == c)
```

Example 2:

```
Input: a = 4, b = 2, c = 7
Output: 1
```

Example 3:

```
Input: a = 1, b = 2, c = 3
Output: 0
```

Constraints:

- `1 <= a <= 10^9`
- `1 <= b <= 10^9`
- `1 <= c <= 10^9`

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

Yes | No

Accepted 177,483/247.3K | Acceptance Rate 71.8%

Topics

187 Online

https://leetcode.com/problems/implement-trie-prefix-tree?envType=study-plan-...

Submit | Premium

Code

Java | Auto

```java
class Solution {
    public int minFlips(int a, int b, int c) {
        int flips = 0;

        while (a > 0 || b > 0 || c > 0) {
            int aBit = a & 1; // last bit of a
            int bBit = b & 1; // last bit of b
            int cBit = c & 1; // last bit of c

            if (cBit == 1) {
                // At least one of aBit or bBit must be 1
                if (aBit == 0 && bBit == 0) {
                    flips += 1;
                }
            } else {
                // Both aBit and bBit must be 0
                if (aBit == 1 && bBit == 1) {
                    flips += 2;
                } else if (aBit == 1 || bBit == 1) {
                    flips += 1;
                }
            }

            a >>= 1; // shift right
            b >>= 1;
            c >>= 1;
        }

        return flips;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

Trie

LeetCode 75

Description | Editorial | Solutions | Submissions

208. Implement Trie (Prefix Tree)

Solved

Medium | Topics | Companies

A trie (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

Example 1:

```
Input
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
Output
[null, null, true, false, true, null, true]

Explanation
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // return True
trie.search("app");     // return False
trie.startsWith("app"); // return True
trie.insert("app");
trie.search("app");     // return True
```

Constraints:

- `1 <= word.length, prefix.length <= 2000`
- `word` and `prefix` consist only of lowercase English letters.
- At most `3 * 10^4` calls in total will be made to `insert`, `search`, and `startsWith`.

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

12.2K | 87 | 99 Online

Submit | Premium

Code

Java | Auto

```java
class TrieNode {
    TrieNode[] children;
    boolean isEndOfWord;

    TrieNode() {
        children = new TrieNode[26]; // For lowercase 'a' to 'z'
        isEndOfWord = false;
    }
}

public class Trie {
    private final TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode curr = root;
        for (char c : word.toCharArray()) {
            int idx = c - 'a';
            if (curr.children[idx] == null) {
                curr.children[idx] = new TrieNode();
            }
            curr = curr.children[idx];
        }
        curr.isEndOfWord = true;
    }

    public boolean search(String word) {
        TrieNode node = searchPrefix(word);
        return node != null && node.isEndOfWord;
    }

    public boolean startsWith(String prefix) {
        return searchPrefix(prefix) != null;
    }

    private TrieNode searchPrefix(String prefix) {
        TrieNode curr = root;
        for (char c : prefix.toCharArray()) {
            int idx = c - 'a';
            if (curr.children[idx] == null) {
                return null;
            }
            curr = curr.children[idx];
        }
        return curr;
    }
```

Saved | Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

1268. Search Suggestions System

Solved

Medium | Topics | Companies | Hint

You are given an array of strings `products` and a string `searchWord`.

Design a system that suggests at most three product names from `products` after each character of `searchWord` is typed. Suggested products should have common prefix with `searchWord`. If there are more than three products with a common prefix return the three lexicographically minimums products.

Return *a list of lists of the suggested products after each character of* `searchWord` *is typed.*

Example 1:

```
Input: products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord = "mouse"
Output: [["mobile","moneypot","monitor"],["mobile","moneypot","monitor"],
["mouse","mousepad"],["mouse","mousepad"],["mouse","mousepad"]]
Explanation: products sorted lexicographically =
["mobile","moneypot","monitor","mouse","mousepad"].
After typing m and mo all products match and we show user ["mobile","moneypot","monitor"].
After typing mou, mous and mouse the system suggests ["mouse","mousepad"].
```

Example 2:

```
Input: products = ["havana"], searchWord = "havana"
Output: [["havana"],["havana"],["havana"],["havana"],["havana"],["havana"]]
Explanation: The only word "havana" will be always suggested while typing the search word.
```

Constraints:

- `1 <= products.length <= 1000`
- `1 <= products[i].length <= 3000`
- `1 <= sum(products[i].length) <= 2 * 10^4`
- All the strings of `products` are **unique**.
- `products[i]` consists of lowercase English letters.
- `1 <= searchWord.length <= 1000`
- `searchWord` consists of lowercase English letters.

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

31 Online

https://leetcode.com/problems/non-overlapping-intervals?envType=study-plan-v...

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.*;

class Solution {
    public List<List<String>> suggestedProducts(String[] products, String searchWord) {
        Arrays.sort(products);
        List<List<String>> result = new ArrayList<>();

        String prefix = "";
        for (char c : searchWord.toCharArray()) {
            prefix += c;
            List<String> suggestions = new ArrayList<>();

            for (String product : products) {
                if (product.startsWith(prefix)) {
                    suggestions.add(product);
                    if (suggestions.size() == 3) break;
                }
            }

            result.add(suggestions);
        }

        return result;
    }
}
```

Saved | Ln 1, Col 1

Testcase | Test Result

Intervals

LeetCode 75

Description | Editorial | Solutions | Submissions

435. Non-overlapping Intervals

Solved

Medium | Topics | Companies

Given an array of intervals `intervals` where `intervals[i] = [start_i, end_i]`, return the *minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.*

Note that intervals which only touch at a point are **non-overlapping**. For example, `[1, 2]` and `[2, 3]` are non-overlapping.

Example 1:

```
Input: intervals = [[1,2],[2,3],[3,4],[1,3]]
Output: 1
Explanation: [1,3] can be removed and the rest of the intervals are non-overlapping.
```

Example 2:

```
Input: intervals = [[1,2],[1,2],[1,2]]
Output: 2
Explanation: You need to remove two [1,2] to make the rest of the intervals non-overlapping.
```

Example 3:

```
Input: intervals = [[1,2],[2,3]]
Output: 0
Explanation: You don't need to remove any of the intervals since they're already non-overlapping.
```

Constraints:

- $1 \le intervals.length \le 10^5$
- $intervals[i].length = 2$
- $-5 \times 10^4 \le start_i < end_i \le 5 \times 10^4$

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

Yes | No

Accepted **842,476** / 1.5M Acceptance Rate **55.9%**

Topics

https://leetcode.com/problems/minimum-number-of-arrows-to-burst-balloons?e...

Submit | Premium

Code

Java | Auto

```java
import java.util.Arrays;

class Solution {
    public int eraseOverlapIntervals(int[][] intervals) {
        if (intervals == null || intervals.length == 0)
            return 0;

        // Sort intervals based on their end time
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[1], b[1]));

        int count = 0;
        int end = intervals[0][1];

        for (int i = 1; i < intervals.length; i++) {
            if (intervals[i][0] < end) {
                // Overlapping interval, increment count
                count++;
            } else {
                // No overlap, update end time
                end = intervals[i][1];
            }
        }

        return count;
    }
}
```

Saved Ln 1, Col 1

Testcase | Test Result

LeetCode 75

Description | Editorial | Solutions | Submissions

452. Minimum Number of Arrows to Burst Balloons

Solved

Medium | Topics | Companies

There are some spherical balloons taped onto a flat wall that represents the XY-plane. The balloons are represented as a 2D integer array `points` where `points[i] = [x_start, x_end]` denotes a balloon whose **horizontal diameter** stretches between x_{start} and x_{end}. You do not know the exact y-coordinates of the balloons.

Arrows can be shot up **directly vertically** (in the positive y-direction) from different points along the x-axis. A balloon with x_{start} and x_{end} is **burst** by an arrow shot at x if $x_{start} \le x \le x_{end}$. There is **no limit** to the number of arrows that can be shot. A shot arrow keeps traveling up infinitely, bursting any balloons in its path.

Given the array `points`, return *the **minimum** number of arrows that must be shot to burst all balloons.*

Example 1:

```
Input: points = [[10,16],[2,8],[1,6],[7,12]]
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6].
- Shoot an arrow at x = 11, bursting the balloons [10,16] and [7,12].
```

Example 2:

```
Input: points = [[1,2],[3,4],[5,6],[7,8]]
Output: 4
Explanation: One arrow needs to be shot for each balloon for a total of 4 arrows.
```

Example 3:

```
Input: points = [[1,2],[2,3],[3,4],[4,5]]
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at x = 2, bursting the balloons [1,2] and [2,3].
- Shoot an arrow at x = 4, bursting the balloons [3,4] and [4,5].
```

Constraints:

- $1 \le points.length \le 10^5$
- $points[i].length = 2$
- $-2^{31} \le x_{start} < x_{end} \le 2^{31} - 1$

https://leetcode.com/problems/daily-temperatures?envType=study-plan-v2&envI...

Submit | Premium

Code

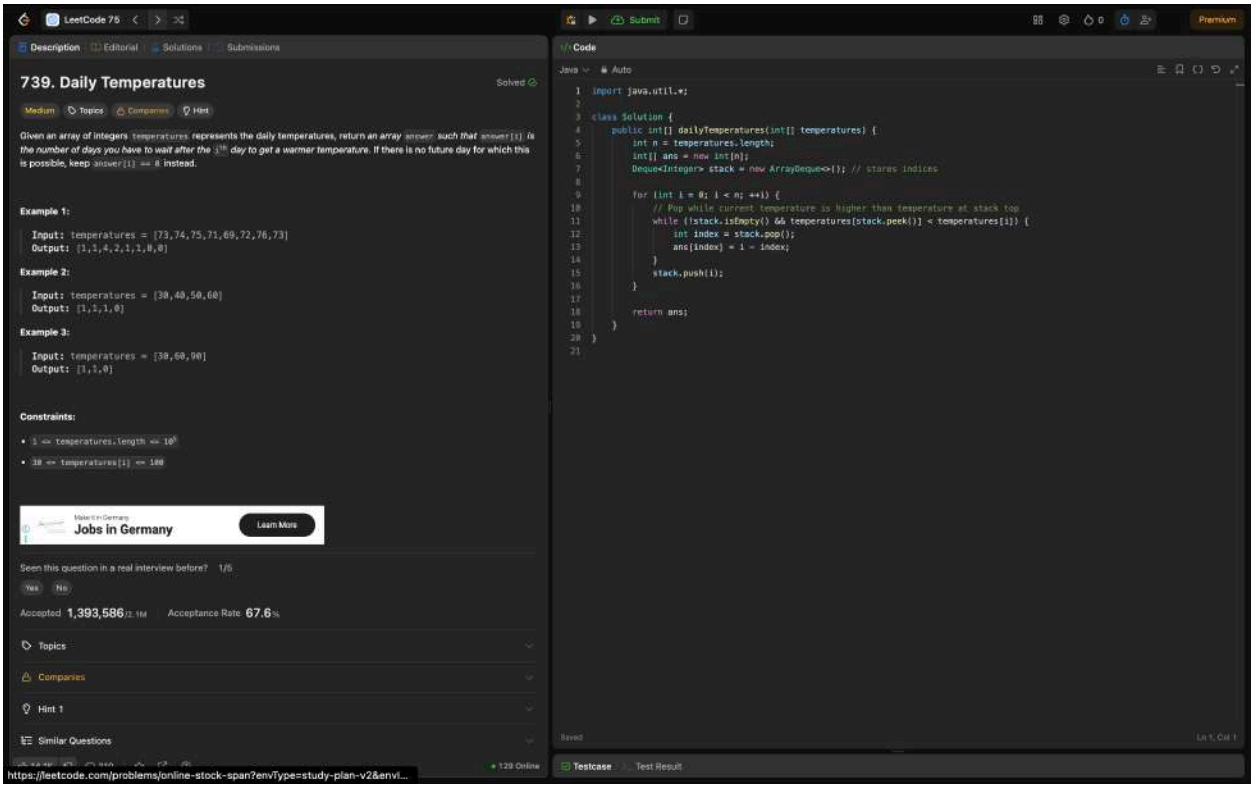
Java | Auto

```java
import java.util.*;

class Solution {
    public int findMinArrowShots(int[][] points) {
        if (points.length == 0) return 0;

        Arrays.sort(points, (a, b) -> Integer.compare(a[1], b[1]));

        int arrows = 1;
        int prevEnd = points[0][1];

        for (int[] balloon : points) {
            if (balloon[0] > prevEnd) {
                arrows++;
                prevEnd = balloon[1];
            }
        }

        return arrows;
    }

    public static void main(String[] args) {
        Solution sol = new Solution();
        int[][] points = {{10,16},{2,8},{1,6},{7,12}};
        System.out.println(sol.findMinArrowShots(points)); // Output: 2
    }
}
```

Saved Ln 1, Col 1

Testcase | Test Result

Monotonic Stack



LeetCode 75

Description | Editorial | Solutions | Submissions

901. Online Stock Span

Solved

Medium | Topics | Companies

Design an algorithm that collects daily price quotes for some stock and returns **the span** of that stock's price for the current day.

The **span** of the stock's price in one day is the maximum number of consecutive days (starting from that day and going backward) for which the stock price was less than or equal to the price of that day.

- For example, if the prices of the stock in the last four days is `[7,2,1,2]` and the price of the stock today is `2`, then the span of today is `4` because starting from today, the price of the stock was less than or equal `2` for `4` consecutive days.
- Also, if the prices of the stock in the last four days is `[7,34,1,2]` and the price of the stock today is `8`, then the span of today is `3` because starting from today, the price of the stock was less than or equal `8` for `3` consecutive days.

Implement the `StockSpanner` class:

- `StockSpanner()` Initializes the object of the class.
- `int next(int price)` Returns the **span** of the stock's price given that today's price is `price`.

Example 1:

```
Input
["StockSpanner", "next", "next", "next", "next", "next", "next", "next"]
[[], [100], [80], [60], [70], [60], [75], [85]]
Output
[null, 1, 1, 1, 2, 1, 4, 6]

Explanation
StockSpanner stockSpanner = new StockSpanner();
stockSpanner.next(100); // return 1
stockSpanner.next(80);  // return 1
stockSpanner.next(60);  // return 1
stockSpanner.next(70);  // return 2
stockSpanner.next(60);  // return 1
stockSpanner.next(75);  // return 4, because the last 4 prices (including today's price of 75) were less than or equal to today's price.
stockSpanner.next(85);  // return 6
```

Constraints:

- $1 \le price \le 10^5$
- At most 10^4 calls will be made to `next`.

59 Online

https://leetcode.com/problems/merge-strings-alternately?envType=study-plan-v...

Submit | 0 | Premium

Code

Java | Auto

```java
import java.util.Stack;

class StockSpanner {
    Stack<int[]> stack;

    public StockSpanner() {
        stack = new Stack<>();
    }

    public int next(int price) {
        int span = 1;
        while (!stack.isEmpty() && stack.peek()[0] <= price) {
            span += stack.pop()[1];
        }
        stack.push(new int[]{price, span});
        return span;
    }
}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * StockSpanner obj = new StockSpanner();
 * int param_1 = obj.next(price);
 */
```

Saved | Ln 1, Col 1

Testcase | Test Result