Blind 75 LeetCode Solutions

Table of Contents

1. Two Sum
2. 3Sum
3. Search in Rotated Sorted Array
4. Product of Array Except Self
5. Maximum Subarray
6. Maximum Product Subarray
7. Find Minimum in Rotated Sorted Array
8. Contains Duplicate
9. Container With Most Water
10. Best Time to Buy and Sell Stock
11. Valid Parentheses
12. Valid Palindrome
13. Valid Anagram
14. Palindromic Substrings
15. Minimum Window Substring
16. Longest Substring Without Repeating Characters
17. Longest Repeating Character Replacement
18. Longest Palindromic Substring
19. Group Anagrams
20. Encode and Decode Strings
21. Sum of Two Integers
22. Reverse Bits
23. Number of 1 Bits
24. Missing Number
25. Counting Bits
26. Reverse Linked List
27. Reorder List
28. Remove Nth Node From End of List
29. Merge Two Sorted Lists
30. Linked List Cycle
31. Intersection of Two Linked Lists
32. Validate Binary Search Tree
33. Subtree of Another Tree
34. Serialize and Deserialize Binary Tree
35. Same Tree
36. Maximum Depth of Binary Tree
37. Lowest Common Ancestor of a Binary Search Tree
38. Kth Smallest Element in a BST
39. Invert Binary Tree
40. Implement Trie (Prefix Tree)
41. Construct Binary Tree from Preorder and Inorder Traversal
42. Binary Tree Maximum Path Sum
43. Binary Tree Level Order Traversal
44. Word Break
45. Unique Paths
46. Longest Increasing Subsequence
47. Jump Game
48. House Robber II
49. House Robber

50. Decode Ways
51. Combination Sum
52. Coin Change
53. Climbing Stairs
54. Pacific Atlantic Water Flow
55. Number of Islands
56. Number of Connected Components in an Undirected Graph
57. Longest Consecutive Sequence
58. Graph Valid Tree
59. Course Schedule
60. Clone Graph
61. Alien Dictionary
62. Non-overlapping Intervals
63. Merge Intervals
64. Meeting Rooms II
65. Meeting Rooms
66. Insert Interval
67. Word Search
68. Spiral Matrix
69. Set Matrix Zeroes
70. Rotate Image
71. Top K Frequent Elements
72. Merge k Sorted Lists
73. Find Median from Data Stream

Two Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9 Output: [0,1] Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6 Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6 Output: [0,1]

Constraints:

```
2 <= nums.length <= 104
-109 <= nums[i] <= 109
-109 <= target <= 109
Only one valid answer exists.
```

https://leetcode.com/problems/two-sum/description/

Easy Solution

```
def easy_solution(nums: List[int], target: int) -> List[int]:
    # Brute-force solution: Check every pair of numbers
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            if nums[i] + nums[j] == target:
                return [i, j]
    return []
```

Optimized Solution

```
def optimized_solution(nums: List[int], target: int) -> List[in
    # Optimized solution: Use a dictionary to track numbers and
    num_dict = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_dict:
```

```
                return [num_dict[complement], i]
            num_dict[num] = i
        return []
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- 3Sum
- 4Sum
- Two Sum II - Input array is sorted

3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4] Output: [[-1,-1,2],[-1,0,1]]

Example 2:

Input: nums = [] Output: []

Example 3:

Input: nums = [0] Output: []

Constraints:

```
0 <= nums.length <= 3000
-105 <= nums[i] <= 105
```

https://leetcode.com/problems/3sum/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> List[List[int]]:
    # Brute-force solution: Check all triplets
    result = []
    nums.sort()
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, len(nums) - 1):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            for k in range(j + 1, len(nums)):
                if k > j + 1 and nums[k] == nums[k - 1]:
                    continue
                if nums[i] + nums[j] + nums[k] == 0:
                    result.append([nums[i], nums[j], nums[k]])
    return result
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> List[List[int]]:
    # Optimized solution: Use two pointers
    result = []
```

```
        nums.sort()
        for i in range(len(nums) - 2):
            if i > 0 and nums[i] == nums[i - 1]:
                continue
            left, right = i + 1, len(nums) - 1
            while left < right:
                total = nums[i] + nums[left] + nums[right]
                if total == 0:
                    result.append([nums[i], nums[left], nums[right]
                    while left < right and nums[left] == nums[left
                        left += 1
                    while left < right and nums[right] == nums[righ
                        right -= 1
                    left += 1
                    right -= 1
                elif total < 0:
                    left += 1
                else:
                    right -= 1
        return result
```

Time Complexity: O(n^2) Space Complexity: O(1)

Similar Questions

- Two Sum
- 3Sum Closest
- 4Sum

Search in Rotated Sorted Array

There is an integer array nums sorted in ascending order (with distinct values).

Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0 Output: 4

Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3 Output: -1

Example 3:

Input: nums = [1], target = 0 Output: -1

Constraints:

```
1 <= nums.length <= 5000
-104 <= nums[i] <= 104
All values of nums are unique.
nums is an ascending array that is possibly rotated.
-104 <= target <= 104
```

https://leetcode.com/problems/search-in-rotated-sorted-array/description/

Easy Solution

```
def easy_solution(nums: List[int], target: int) -> int:
    # Brute-force solution: Use built-in index function
    return nums.index(target) if target in nums else -1
```

Optimized Solution

```
def optimized_solution(nums: List[int], target: int) -> int:
    # Optimized solution: Use binary search
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = left + (right - left) // 2
        if nums[mid] == target:
            return mid
```

```
            if nums[left] <= nums[mid]:
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] < target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

Time Complexity: O(log n) Space Complexity: O(1)

Similar Questions

- Search in Rotated Sorted Array II
- Find Minimum in Rotated Sorted Array

Product of Array Except Self

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 <= 105
−30 <= nums[i] <= 30
The product of any prefix or suffix of nums is guaranteed to fit in a 3
```

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.)

https://leetcode.com/problems/product-of-array-except-self/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> List[int]:
    # Brute-force solution: Calculate product for each index
    n = len(nums)
    result = [1] * n
    for i in range(n):
        for j in range(n):
            if i != j:
                result[i] *= nums[j]
    return result
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> List[int]:
    # Optimized solution: Use left and right product arrays
    n = len(nums)
    result = [1] * n
    left_product = 1
    right_product = 1
    for i in range(n):
        result[i] *= left_product
```

```
        left_product *= nums[i]
        result[n - 1 - i] *= right_product
        right_product *= nums[n - 1 - i]
    return result
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Trapping Rain Water
- Maximum Product Subarray

Maximum Subarray

Given an integer array nums, find the subarray with the largest sum, and return its sum.

A subarray is a contiguous part of an array.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4] Output: 6 Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input: nums = [1] Output: 1

Example 3:

Input: nums = [5,4,-1,7,8] Output: 23

Constraints:

```
1 <= nums.length <= 105
−104 <= nums[i] <= 104
```

https://leetcode.com/problems/maximum-subarray/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> int:
    # Brute-force solution: Check all subarrays
    max_sum = float('-inf')
    for i in range(len(nums)):
        current_sum = 0
        for j in range(i, len(nums)):
            current_sum += nums[j]
            max_sum = max(max_sum, current_sum)
    return max_sum
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> int:
    # Optimized solution: Use Kadane's algorithm
    max_sum = current_sum = nums[0]
    for num in nums[1:]:
        current_sum = max(num, current_sum + num)
        max_sum = max(max_sum, current_sum)
    return max_sum
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Best Time to Buy and Sell Stock
- Maximum Product Subarray

Maximum Product Subarray

Given an integer array nums, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example 1:

Input: nums = [2,3,-2,4] Output: 6 Explanation: [2,3] has the largest product 6.

Example 2:

Input: nums = [-2,0,-1] Output: 0 Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Constraints:

```
1 <= nums.length <= 105
−10 <= nums[i] <= 10
The product of any prefix or suffix of nums is guaranteed to fit in a 3
```

https://leetcode.com/problems/maximum-product-subarray/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> int:
    # Brute-force solution: Check all subarrays
    max_product = float('-inf')
    for i in range(len(nums)):
        current_product = 1
        for j in range(i, len(nums)):
            current_product *= nums[j]
            max_product = max(max_product, current_product)
    return max_product
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> int:
    # Optimized solution: Track max and min products
    max_so_far = min_so_far = result = nums[0]
    for i in range(1, len(nums)):
        temp_max = max(nums[i], max_so_far * nums[i], min_so_fa
        min_so_far = min(nums[i], max_so_far * nums[i], min_so_
        max_so_far = temp_max
        result = max(result, max_so_far)
    return result
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Maximum Subarray
- House Robber

Find Minimum in Rotated Sorted Array

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

[4,5,6,7,0,1,2] if it was rotated 4 times. [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return the minimum element of this array.

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

Example 1:

Input: nums = [3,4,5,1,2] Output: 1 Explanation: The original array was [1,2,3,4,5] rotated 3 times.

Example 2:

Input: nums = [4,5,6,7,0,1,2] Output: 0 Explanation: The original array was [0,1,2,4,5,6,7] and it was rotated 4 times.

Example 3:

Input: nums = [11,13,15,17] Output: 11 Explanation: The original array was [11,13,15,17] and it was rotated 4 times.

Constraints:

```
n == nums.length
1 <= n <= 5000
-5000 <= nums[i] <= 5000
All the integers of nums are unique.
nums is guaranteed to be rotated at some pivot.
```

https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> int:
    # Brute-force solution: Use built-in min function
    return min(nums)
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> int:
    # Optimized solution: Use binary search
    left, right = 0, len(nums) - 1
```

```
        while left < right:
            mid = left + (right - left) // 2
            if nums[mid] > nums[right]:
                left = mid + 1
            else:
                right = mid
        return nums[left]
```

Time Complexity: O(log n) Space Complexity: O(1)

Similar Questions

- Search in Rotated Sorted Array
- Find Minimum in Rotated Sorted Array II

Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct. Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:

Input: nums = [1,2,3,1] Output: true

Example 2:

Input: nums = [1,2,3,4] Output: false

Example 3:

Input: nums = [1,1,1,3,3,4,3,2,4,2] Output: true

Constraints:

```
1 <= nums.length <= 105
-109 <= nums[i] <= 109
```

https://leetcode.com/problems/contains-duplicate/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> bool:
    # Brute-force solution: Check every pair of numbers
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            if nums[i] == nums[j]:
                return True
    return False
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> bool:
    # Optimized solution: Use a set to track seen numbers
    return len(nums) != len(set(nums))
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Contains Duplicate II
- Contains Duplicate III

Container With Most Water

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.

Example 2:

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

Constraints:

```
n == height.length
2 <= n <= 105
0 <= height[i] <= 104
```

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

Easy Solution

```
def easy_solution(height: List[int]) -> int:
    # Brute-force solution: Check all pairs of lines
    max_area = 0
    for i in range(len(height)):
        for j in range(i + 1, len(height)):
            area = min(height[i], height[j]) * (j - i)
            max_area = max(max_area, area)
    return max_area
```

Optimized Solution

```
def optimized_solution(height: List[int]) -> int:
    # Optimized solution: Use two pointers
    max_area = 0
    left, right = 0, len(height) - 1
    while left < right:
        area = min(height[left], height[right]) * (right - left
        max_area = max(max_area, area)
```

```
            if height[left] < height[right]:
                left += 1
            else:
                right -= 1
        return max_area
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Trapping Rain Water

Best Time to Buy and Sell Stock

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1:

Input: prices = [7,1,5,3,6,4] Output: 5 Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5. Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: prices = [7,6,4,3,1] Output: 0 Explanation: In this case, no transactions are done and the max profit = 0.

Constraints:

```
1 <= prices.length <= 105
0 <= prices[i] <= 104
```

https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/

Easy Solution

```
def easy_solution(prices: List[int]) -> int:
    # Brute-force solution: Check every pair of days
    max_profit = 0
    for i in range(len(prices)):
        for j in range(i + 1, len(prices)):
            profit = prices[j] - prices[i]
            if profit > max_profit:
                max_profit = profit
    return max_profit
```

Optimized Solution

```
def optimized_solution(prices: List[int]) -> int:
    # Optimized solution: Track the minimum price and maximum p
    if not prices:
        return 0
    max_profit = 0
    min_price = float('inf')
    for price in prices:
        if price < min_price:
```

```
            min_price = price
        else:
            max_profit = max(max_profit, price - min_price)
    return max_profit
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Best Time to Buy and Sell Stock II
- Best Time to Buy and Sell Stock III

Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if: Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order. Every close bracket has a corresponding open bracket of the same type.

Example 1: Input: s = "()" Output: true

Example 2: Input: s = "()[]{}" Output: true

Example 3: Input: s = "(]" Output: false

Constraints: 1 <= s.length <= 104 s consists of parentheses only '()[]{}'.

https://leetcode.com/problems/valid-parentheses/description/

Easy Solution

```
def easy_solution(s: str) -> bool:
    # Brute-force solution: Use stack to check validity
    stack = []
    for char in s:
        if char in "({[":
            stack.append(char)
        elif char in ")}]":
            if not stack:
                return False
            if char == ")" and stack[-1] != "(":
                return False
            if char == "}" and stack[-1] != "{":
                return False
            if char == "]" and stack[-1] != "[":
                return False
            stack.pop()
    return len(stack) == 0
```

Optimized Solution

```
def optimized_solution(s: str) -> bool:
    # Optimized solution: Use hashmap for mapping and stack
    stack = []
    mapping = {")": "(", "}": "{", "]": "["}
    for char in s:
        if char in mapping:
            top_element = stack.pop() if stack else '#'
            if mapping[char] != top_element:
                return False
```

```
            else:
                stack.append(char)
        return not stack
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Generate Parentheses
- Longest Valid Parentheses
- Remove Invalid Parentheses

Valid Palindrome

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

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

Example 1: Input: s = "A man, a plan, a canal: Panama" Output: true Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2: Input: s = "race a car" Output: false Explanation: "raceacar" is not a palindrome.

Constraints: 1 <= s.length <= 2 * 105 s consists only of printable ASCII characters.

https://leetcode.com/problems/valid-palindrome/description/

Easy Solution

```
def easy_solution(s: str) -> bool:
    # Brute-force solution: Clean string and check palindrome
    cleaned = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned == cleaned[::-1]
```

Optimized Solution

```
def optimized_solution(s: str) -> bool:
    # Optimized solution: Use two pointers
    left, right = 0, len(s) - 1
    while left < right:
        while left < right and not s[left].isalnum():
            left += 1
        while left < right and not s[right].isalnum():
            right -= 1
        if s[left].lower() != s[right].lower():
            return False
        left += 1
        right -= 1
    return True
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Palindrome Linked List
- Valid Palindrome II

Valid Anagram

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1: Input: s = "anagram", t = "nagaram" Output: true

Example 2: Input: s = "rat", t = "car" Output: false

Constraints: 1 <= s.length, t.length <= 5 * 104 s and t consist of lowercase English letters.

Follow up: What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

https://leetcode.com/problems/valid-anagram/description/

Easy Solution

```
def easy_solution(s: str, t: str) -> bool:
    # Brute-force solution: Sort and compare
    return sorted(s) == sorted(t)
```

Optimized Solution

```
def optimized_solution(s: str, t: str) -> bool:
    # Optimized solution: Use hashmap to count characters
    if len(s) != len(t):
        return False
    char_count = {}
    for c in s:
        char_count[c] = char_count.get(c, 0) + 1
    for c in t:
        if c not in char_count:
            return False
        char_count[c] -= 1
        if char_count[c] == 0:
            del char_count[c]
    return len(char_count) == 0
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Group Anagrams
- Palindrome Permutation

Palindromic Substrings

Given a string s, return the number of palindromic substrings in it.

A string is a palindrome when it reads the same backward as forward.

A substring is a contiguous sequence of characters within the string.

Example 1: Input: s = "abc" Output: 3 Explanation: Three palindromic strings: "a", "b", "c".

Example 2: Input: s = "aaa" Output: 6 Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".

Constraints: 1 <= s.length <= 1000 s consists of lowercase English letters.

https://leetcode.com/problems/palindromic-substrings/description/

Easy Solution

```
def easy_solution(s: str) -> int:
    # Brute-force solution: Check all substrings
    def is_palindrome(sub: str) -> bool:
        return sub == sub[::-1]

    count = 0
    for i in range(len(s)):
        for j in range(i, len(s)):
            if is_palindrome(s[i:j+1]):
                count += 1
    return count
```

Optimized Solution

```
def optimized_solution(s: str) -> int:
    # Optimized solution: Expand around center
    def count_palindromes(left: int, right: int) -> int:
        count = 0
        while left >= 0 and right < len(s) and s[left] == s[rig
            count += 1
            left -= 1
            right += 1
        return count

    total_count = 0
    for i in range(len(s)):
        total_count += count_palindromes(i, i)  # Odd length pa
        total_count += count_palindromes(i, i+1)  # Even length
    return total_count
```

Time Complexity: O(n^2) Space Complexity: O(1)

Similar Questions

- Longest Palindromic Substring
- Longest Palindromic Subsequence

Minimum Window Substring

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string "".

The testcases will be generated such that the answer is unique.

Example 1: Input: s = "ADOBECODEBANC", t = "ABC" Output: "BANC" Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2: Input: s = "a", t = "a" Output: "a" Explanation: The entire string s is the minimum window.

Example 3: Input: s = "a", t = "aa" Output: "" Explanation: Both 'a's from t must be included in the window. Since the largest window of s only has one 'a', return empty string.

Constraints: m == s.length n == t.length 1 <= m, n <= 105 s and t consist of uppercase and lowercase English letters.

Follow up: Could you find an algorithm that runs in O(m + n) time?

https://leetcode.com/problems/minimum-window-substring/description/

Easy Solution

```
def easy_solution(s: str, t: str) -> str:
    # Brute-force solution: Check all substrings
    def contains_all(window: str, target: str) -> bool:
        return all(window.count(c) >= target.count(c) for c in

    min_window = ""
    min_length = float('inf')
    for i in range(len(s)):
        for j in range(i, len(s)):
            window = s[i:j+1]
            if contains_all(window, t) and len(window) < min_le
                min_window = window
                min_length = len(window)
    return min_window
```

Optimized Solution

```
def optimized_solution(s: str, t: str) -> str:
    # Optimized solution: Use sliding window and hashmap
    if not t or not s:
        return ""

    dict_t = {}
    for c in t:
```

```
        dict_t[c] = dict_t.get(c, 0) + 1

    required = len(dict_t)
    left = right = 0
    formed = 0
    window_counts = {}
    ans = float("inf"), None, None

    while right < len(s):
        character = s[right]
        window_counts[character] = window_counts.get(character,

        if character in dict_t and window_counts[character] ==
            formed += 1

        while left <= right and formed == required:
            character = s[left]

            if right - left + 1 < ans[0]:
                ans = (right - left + 1, left, right)

            window_counts[character] -= 1
            if character in dict_t and window_counts[character]
                formed -= 1

            left += 1

        right += 1

    return "" if ans[0] == float("inf") else s[ans[1] : ans[2]
```

Time Complexity: O(n) Space Complexity: O(k)

Similar Questions

- Substring with Concatenation of All Words
- Minimum Size Subarray Sum
- Sliding Window Maximum

Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

Example 1: Input: s = "abcabcbb" Output: 3 Explanation: The answer is "abc", with the length of 3.

Example 2: Input: s = "bbbbb" Output: 1 Explanation: The answer is "b", with the length of 1.

Example 3: Input: s = "pwwkew" Output: 3 Explanation: The answer is "wke", with the length of 3. Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints: 0 <= s.length <= 5 * 104 s consists of English letters, digits, symbols and spaces.

https://leetcode.com/problems/longest-substring-without-repeating-characters/description/

Easy Solution

```
def easy_solution(s: str) -> int:
    # Brute-force solution: Check all substrings
    max_length = 0
    for i in range(len(s)):
        seen = set()
        for j in range(i, len(s)):
            if s[j] in seen:
                break
            seen.add(s[j])
        max_length = max(max_length, len(seen))
    return max_length
```

Optimized Solution

```
def optimized_solution(s: str) -> int:
    # Optimized solution: Use sliding window and hashmap
    char_index = {}
    max_length = start = 0
    for i, char in enumerate(s):
        if char in char_index and start <= char_index[char]:
            start = char_index[char] + 1
        else:
            max_length = max(max_length, i - start + 1)
        char_index[char] = i
    return max_length
```

Time Complexity: O(n) Space Complexity: O(min(m, n))

Similar Questions

- Longest Substring with At Most Two Distinct Characters
- Longest Substring with At Most K Distinct Characters

Longest Repeating Character Replacement

You are given a string s and an integer k. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most k times.

Return the length of the longest substring containing the same letter you can get after performing the above operations.

Example 1: Input: s = "ABAB", k = 2 Output: 4 Explanation: Replace the two 'A's with two 'B's or vice versa.

Example 2: Input: s = "AABABBA", k = 1 Output: 4 Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".

Constraints: 1 <= s.length <= 105 s consists of only uppercase English letters. 0 <= k <= s.length

https://leetcode.com/problems/longest-repeating-character-replacement/description/

Easy Solution

```
def easy_solution(s: str, k: int) -> int:
    # Brute-force solution: Check all substrings
    max_length = 0
    for i in range(len(s)):
        for j in range(i, len(s)):
            substring = s[i:j+1]
            most_common = max(substring.count(c) for c in set(s
            if len(substring) - most_common <= k:
                max_length = max(max_length, len(substring))
    return max_length
```

Optimized Solution

```
def optimized_solution(s: str, k: int) -> int:
    # Optimized solution: Use sliding window and hashmap
    char_count = {}
    max_length = max_count = start = 0
    for end, char in enumerate(s):
        char_count[char] = char_count.get(char, 0) + 1
        max_count = max(max_count, char_count[char])
        if end - start + 1 - max_count > k:
            char_count[s[start]] -= 1
            start += 1
        max_length = max(max_length, end - start + 1)
    return max_length
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Longest Substring with At Most K Distinct Characters
- Max Consecutive Ones III

Longest Palindromic Substring

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

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

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

Example 3: Input: s = "a" Output: "a"

Example 4: Input: s = "ac" Output: "a"

Constraints: 1 <= s.length <= 1000 s consist of only digits and English letters.

https://leetcode.com/problems/longest-palindromic-substring/description/

Easy Solution

```
def easy_solution(s: str) -> str:
    # Brute-force solution: Check all substrings
    def is_palindrome(sub: str) -> bool:
        return sub == sub[::-1]

    longest = ""
    for i in range(len(s)):
        for j in range(i, len(s)):
            substring = s[i:j+1]
            if is_palindrome(substring) and len(substring) > le
                longest = substring
    return longest
```

Optimized Solution

```
def optimized_solution(s: str) -> str:
    # Optimized solution: Expand around center
    def expand_around_center(left: int, right: int) -> str:
        while left >= 0 and right < len(s) and s[left] == s[rig
            left -= 1
            right += 1
        return s[left+1:right]

    if len(s) < 2:
        return s

    longest = ""
    for i in range(len(s)):
        palindrome1 = expand_around_center(i, i)
        palindrome2 = expand_around_center(i, i+1)
        longest = max(longest, palindrome1, palindrome2, key=le
    return longest
```

Time Complexity: O(n^2) Space Complexity: O(1)

Similar Questions

- Shortest Palindrome
- Palindrome Permutation
- Palindrome Pairs

Group Anagrams

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1: Input: strs = ["eat","tea","tan","ate","nat","bat"] Output: [["bat"],["nat","tan"], ["ate","eat","tea"]]

Example 2: Input: strs = [""] Output: [[""]]

Example 3: Input: strs = ["a"] Output: [["a"]]

Constraints: 1 <= strs.length <= 104 0 <= strs[i].length <= 100 strs[i] consists of lowercase English letters.

https://leetcode.com/problems/group-anagrams/description/

Easy Solution

```
def easy_solution(strs: List[str]) -> List[List[str]]:
    # Brute-force solution: Sort strings and group
    anagram_groups = {}
    for s in strs:
        sorted_s = ''.join(sorted(s))
        if sorted_s in anagram_groups:
            anagram_groups[sorted_s].append(s)
        else:
            anagram_groups[sorted_s] = [s]
    return list(anagram_groups.values())
```

Optimized Solution

```
def optimized_solution(strs: List[str]) -> List[List[str]]:
    # Optimized solution: Use character count as key
    anagram_groups: Dict[str, List[str]] = {}
    for s in strs:
        count = [0] * 26
        for c in s:
            count[ord(c) - ord('a')] += 1
        key = tuple(count)
        if key in anagram_groups:
            anagram_groups[key].append(s)
        else:
            anagram_groups[key] = [s]
    return list(anagram_groups.values())
```

Time Complexity: O(n * k) Space Complexity: O(n * k)

Similar Questions

- Valid Anagram
- Group Shifted Strings

Encode and Decode Strings

Design an algorithm to encode a list of strings to a string. The encoded string is then sent over the network and is decoded back to the original list of strings.

Implement the encode and decode methods.

Example: Input: ["Hello","World"] Output: ["Hello","World"]

Note:

1. The string may contain any possible characters out of 256 valid ascii characters. Your algorithm should be generalized enough to work on any possible characters.
2. Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.

https://leetcode.com/problems/encode-and-decode-strings/description/

Easy Solution

```
def easy_solution(strs: List[str]) -> List[str]:
    # Utilize the Codec class for encoding and decoding
    codec = Codec()
    encoded = codec.encode(strs)
    return codec.decode(encoded)
```

Optimized Solution

```
def optimized_solution(strs: List[str]) -> List[str]:
    # The easy solution is already optimal
    return easy_solution(strs)
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Count and Say
- Serialize and Deserialize Binary Tree

Sum of Two Integers

Given two integers a and b, return the sum of the two integers without using the operators + and -.

Example 1:

Input: a = 1, b = 2 Output: 3

Example 2:

Input: a = 2, b = 3 Output: 5

Constraints:

`-1000 <= a, b <= 1000`

https://leetcode.com/problems/sum-of-two-integers/

Easy Solution

```
def easy_solution(a: int, b: int) -> int:
    # Iterative solution: Use bit manipulation to add without +
    while b != 0:
        carry = a & b
        a = a ^ b
        b = carry << 1
    return a
```

Optimized Solution

```
def optimized_solution(a: int, b: int) -> int:
    # The easy solution is already optimal for this problem
    return easy_solution(a, b)
```

Time Complexity: O(1) Space Complexity: O(1)

Similar Questions

- Add Binary

Reverse Bits

Reverse bits of a given 32 bits unsigned integer.

Note:

```
Note that in some languages, such as Java, there is no unsigned integer
In Java, the compiler represents the signed integers using 2's compleme
```

Example 1:

Input: n = 00000010100101000001111010011100 Output: 964176192 (00111001011110000010100101000000) Explanation: The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 00111001011110000010100101000000.

Example 2:

Input: n = 11111111111111111111111111111101 Output: 3221225471 (10111111111111111111111111111111) Explanation: The input binary string 11111111111111111111111111111101 represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is 10111111111111111111111111111111.

Constraints:

```
The input must be a binary string of length 32
```

https://leetcode.com/problems/reverse-bits/

Easy Solution

```
def easy_solution(n: int) -> int:
    # Iterative solution: Reverse bits by shifting
    result = 0
    for i in range(32):
        result <<= 1
        result |= n & 1
        n >>= 1
    return result
```

Optimized Solution

```
def optimized_solution(n: int) -> int:
    # The easy solution is already optimal for this problem
    return easy_solution(n)
```

Time Complexity: O(1) Space Complexity: O(1)

Similar Questions

- Number of 1 Bits

Number of 1 Bits

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Example 1:

Input: n = 00000000000000000000000000001011 Output: 3 Explanation: The input binary string 00000000000000000000000000001011 has a total of three '1' bits.

Example 2:

Input: n = 00000000000000000000000010000000 Output: 1 Explanation: The input binary string 00000000000000000000000010000000 has a total of one '1' bit.

Example 3:

Input: n = 11111111111111111111111111111101 Output: 31 Explanation: The input binary string 11111111111111111111111111111101 has a total of thirty-one '1' bits.

Constraints:

`The input must be a binary string of length 32.`

https://leetcode.com/problems/number-of-1-bits/

Easy Solution

```
def easy_solution(n: int) -> int:
    # Iterative solution: Count bits by shifting
    count = 0
    while n:
        count += n & 1
        n >>= 1
    return count
```

Optimized Solution

```
def optimized_solution(n: int) -> int:
    # Optimized solution: Use n & (n-1) to count bits
    count = 0
    while n:
        n &= n - 1
        count += 1
    return count
```

Time Complexity: O(1) Space Complexity: O(1)

Similar Questions

- Hamming Distance
- Reverse Bits

Missing Number

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Example 1:

Input: nums = [3,0,1] Output: 2 Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0,1] Output: 2 Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1] Output: 8 Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

Constraints:

```
n == nums.length
1 <= n <= 104
0 <= nums[i] <= n
All the numbers of nums are unique.
```

https://leetcode.com/problems/missing-number/

Easy Solution

```
def easy_solution(nums: List[int]) -> int:
    # Iterative solution: Use summation formula
    n = len(nums)
    total_sum = n * (n + 1) // 2
    return total_sum - sum(nums)
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> int:
    # Optimized solution: Use XOR
    missing = len(nums)
    for i, num in enumerate(nums):
        missing ^= i ^ num
    return missing
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Find the Duplicate Number
- Find All Numbers Disappeared in an Array

Counting Bits

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 <= 105
```

https://leetcode.com/problems/counting-bits/

Easy Solution

```
def easy_solution(n: int) -> List[int]:
    # Iterative solution: Count bits for each number
    def count_bits(x: int) -> int:
        count = 0
        while x:
            count += x & 1
            x >>= 1
        return count

    return [count_bits(i) for i in range(n + 1)]
```

Optimized Solution

```
def optimized_solution(n: int) -> List[int]:
    # Optimized solution: Use dynamic programming
    dp = [0] * (n + 1)
    for i in range(1, n + 1):
        dp[i] = dp[i >> 1] + (i & 1)
    return dp
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Number of 1 Bits
- Binary Watch

Reverse Linked List

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]

Example 3:

Input: head = [] Output: []

Constraints:

```
The number of nodes in the list is the range [0, 5000].
−5000 <= Node.val <= 5000
```

https://leetcode.com/problems/reverse-linked-list/description/

Easy Solution

```
def easy_solution(head: Optional[ListNode]) -> Optional[ListNod
    # Iterative approach: Reverse the list by changing the next
    prev = None
    current = head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev
```

Optimized Solution

```
def optimized_solution(head: Optional[ListNode]) -> Optional[Li
    # The easy solution is already optimal for this problem
    return easy_solution(head)
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Reverse Linked List II
- Binary Tree Upside Down
- Palindrome Linked List

Reorder List

You are given the head of a singly linked-list. The list can be represented as:

L0 → L1 → … → Ln - 1 → Ln

Reorder the list to be on the following form:

L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:

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

Example 2:

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

Constraints:

```
The number of nodes in the list is in the range [1, 5 * 104].
1 <= Node.val <= 1000
```

https://leetcode.com/problems/reorder-list/description/

Easy Solution

```
        def easy_solution(head: Optional[ListNode]) -> None:
            # Optimal approach: Find middle, reverse second half, and m
            if not head or not head.next:
                return

            # Find the middle of the list
            slow = fast = head
            while fast.next and fast.next.next:
                slow = slow.next
                fast = fast.next.next

            # Reverse the second half of the list
            second = slow.next
            slow.next = None
            prev = None
            while second:
                next_node = second.next
                second.next = prev
                prev = second
                second = next_node

            # Merge the two halves
            first = head
```

```
        second = prev
        while second:
            next_first = first.next
            next_second = second.next
            first.next = second
            second.next = next_first
            first = next_first
            second = next_second
```

Optimized Solution

```
    def optimized_solution(head: Optional[ListNode]) -> None:
        # The easy solution is already optimal for this problem
        easy_solution(head)
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Palindrome Linked List

Remove Nth Node From End of List

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Example 1:

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

Example 2:

Input: head = [1], n = 1 Output: []

Example 3:

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

Constraints:

```
The number of nodes in the list is sz.
1 <= sz <= 30
0 <= Node.val <= 100
1 <= n <= sz
```

https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/

Easy Solution

```
        def easy_solution(head: Optional[ListNode], n: int) -> Optional
            # Two-pointer approach: Remove the nth node from the end
            dummy = ListNode(0)
            dummy.next = head
            first = dummy
            second = dummy
            for _ in range(n + 1):
                first = first.next
            while first:
                first = first.next
                second = second.next
            second.next = second.next.next
            return dummy.next
```

Optimized Solution

```
        def optimized_solution(head: Optional[ListNode], n: int) -> Opt
            # The easy solution is already optimal for this problem
            return easy_solution(head, n)
```

Time Complexity: O(L) Space Complexity: O(1)

Similar Questions

- Swapping Nodes in a Linked List
- Delete N Nodes After M Nodes of a Linked List

Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:

Input: list1 = [1,2,4], list2 = [1,3,4] Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = [] Output: []

Example 3:

Input: list1 = [], list2 = [0] Output: [0]

Constraints:

```
The number of nodes in both lists is in the range [0, 50].
-100 <= Node.val <= 100
Both list1 and list2 are sorted in non-decreasing order.
```

https://leetcode.com/problems/merge-two-sorted-lists/description/

Easy Solution

```
        def easy_solution(l1: Optional[ListNode], l2: Optional[ListNode
            # Iterative approach: Merge two sorted lists
            dummy = ListNode(0)
            current = dummy
            while l1 and l2:
                if l1.val <= l2.val:
                    current.next = l1
                    l1 = l1.next
                else:
                    current.next = l2
                    l2 = l2.next
                current = current.next
            current.next = l1 if l1 else l2
            return dummy.next
```

Optimized Solution

```
        def optimized_solution(l1: Optional[ListNode], l2: Optional[Lis
            # The easy solution is already optimal for this problem
```

```
            return easy_solution(l1, l2)
```

Time Complexity: O(n + m) Space Complexity: O(1)

Similar Questions

- Merge k Sorted Lists
- Merge Sorted Array
- Sort List
- Shortest Word Distance II

Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.

Example 1:

Input: head = [3,2,0,-4], pos = 1 Output: true Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:

Input: head = [1,2], pos = 0 Output: true Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:

Input: head = [1], pos = -1 Output: false Explanation: There is no cycle in the linked list.

Constraints:

```
The number of the nodes in the list is in the range [0, 104].
-105 <= Node.val <= 105
pos is -1 or a valid index in the linked list.
```

https://leetcode.com/problems/linked-list-cycle/description/

Easy Solution

```
        def easy_solution(head: Optional[ListNode]) -> bool:
            # Brute-force solution: Use a set to track visited nodes
            seen = set()
            current = head
            while current:
                if current in seen:
                    return True
                seen.add(current)
                current = current.next
            return False
```

Optimized Solution

```
        def optimized_solution(head: Optional[ListNode]) -> bool:
            # Optimized solution: Use two pointers (Floyd's Tortoise an
            if not head or not head.next:
```

```
        return False
    slow = head
    fast = head.next
    while slow != fast:
        if not fast or not fast.next:
            return False
        slow = slow.next
        fast = fast.next.next
    return True
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Linked List Cycle II
- Happy Number

Intersection of Two Linked Lists

Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.

For example, the following two linked lists begin to intersect at node c1:

Example 1:

Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3 Output: Intersected at '8' Explanation: The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

Example 2:

Input: intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1 Output: Intersected at '2' Explanation: The intersected node's value is 2 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [1,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1 node before the intersected node in B.

Example 3:

Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2 Output: No intersection Explanation: From the head of A, it reads as [2,6,4]. From the head of B, it reads as [1,5]. Since the two lists do not intersect, intersectVal must be 0.

Constraints:

```
The number of nodes of listA is in the m.
The number of nodes of listB is in the n.
1 <= m, n <= 3 * 104
1 <= Node.val <= 105
0 <= skipA < m
0 <= skipB < n
intersectVal is 0 if listA and listB do not intersect.
intersectVal == listA[skipA] == listB[skipB] if listA and listB interse
```

https://leetcode.com/problems/intersection-of-two-linked-lists/description/

Easy Solution

```
        def easy_solution(headA: ListNode, headB: ListNode) -> Optional
            # Two-pointer approach: Find intersection node
            if not headA or not headB:
                return None

            nodeA = headA
            nodeB = headB
```

```
        while nodeA != nodeB:
            nodeA = nodeA.next if nodeA else headB
            nodeB = nodeB.next if nodeB else headA

        return nodeA
```

Optimized Solution

```
    def optimized_solution(headA: ListNode, headB: ListNode) -> Opt
        # The easy solution is already optimal for this problem
        return easy_solution(headA, headB)
```

Time Complexity: O(n + m) Space Complexity: O(1)

Similar Questions

- Minimum Index Sum of Two Lists

Validate Binary Search Tree

Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A valid BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees.

Example 1:

Input: root = [2,1,3] Output: true

Example 2:

Input: root = [5,1,4,null,null,3,6] Output: false Explanation: The root node's value is 5 but its right child's value is 4.

Constraints:

```
The number of nodes in the tree is in the range [1, 104].
-231 <= Node.val <= 231 - 1
```

https://leetcode.com/problems/validate-binary-search-tree/

Easy Solution

```
        def easy_solution(root: Optional[TreeNode]) -> bool:
            # Recursive solution: Check each node's value within valid
            def is_valid_bst(node: Optional[TreeNode], min_val: float,
                if not node:
                    return True
                if node.val <= min_val or node.val >= max_val:
                    return False
                return (is_valid_bst(node.left, min_val, node.val) and
                        is_valid_bst(node.right, node.val, max_val))

            return is_valid_bst(root, float('-inf'), float('inf'))
```

Optimized Solution

```
        def optimized_solution(root: Optional[TreeNode]) -> bool:
            # Iterative solution: Use a stack to check nodes
            stack = [(root, float('-inf'), float('inf'))]
            while stack:
                node, min_val, max_val = stack.pop()
                if not node:
                    continue
                if node.val <= min_val or node.val >= max_val:
```

```
                return False
            stack.append((node.right, node.val, max_val))
            stack.append((node.left, min_val, node.val))
        return True
```

Time Complexity: O(n) Space Complexity: O(h)

Similar Questions

- Binary Tree Inorder Traversal
- Find Mode in Binary Search Tree

Subtree of Another Tree

Given the roots of two binary trees root and subRoot, return true if there is a subtree of root with the same structure and node values of subRoot and false otherwise.

A subtree of a binary tree tree is a tree that consists of a node in tree and all of this node's descendants. The tree tree could also be considered as a subtree of itself.

Example 1:

Input: root = [3,4,5,1,2], subRoot = [4,1,2] Output: true

Example 2:

Input: root = [3,4,5,1,2,null,null,null,null,0], subRoot = [4,1,2] Output: false

Constraints:

```
The number of nodes in the root tree is in the range [1, 2000].
The number of nodes in the subRoot tree is in the range [1, 1000].
-104 <= root.val <= 104
-104 <= subRoot.val <= 104
```

https://leetcode.com/problems/subtree-of-another-tree/

Easy Solution

```
        def easy_solution(root: Optional[TreeNode], subRoot: Optional[T
            # Recursive solution: Check each subtree
            if not root:
                return False
            if is_same_tree(root, subRoot):
                return True
            return easy_solution(root.left, subRoot) or easy_solution(r
```

Optimized Solution

```
        def optimized_solution(root: Optional[TreeNode], subRoot: Optio
            # Optimized solution: Serialize trees and use KMP algorithm
            def serialize(node: Optional[TreeNode]) -> str:
                if not node:
                    return "#"
                return f"{node.val},{serialize(node.left)},{serialize(n

            def kmp_search(text: str, pattern: str) -> bool:
                if not pattern:
                    return True
                lps = [0] * len(pattern)
                length = 0
                i = 1
```

```
            while i < len(pattern):
                if pattern[i] == pattern[length]:
                    length += 1
                    lps[i] = length
                    i += 1
                elif length != 0:
                    length = lps[length - 1]
                else:
                    lps[i] = 0
                    i += 1

            i = j = 0
            while i < len(text):
                if pattern[j] == text[i]:
                    i += 1
                    j += 1
                if j == len(pattern):
                    return True
                elif i < len(text) and pattern[j] != text[i]:
                    if j != 0:
                        j = lps[j - 1]
                    else:
                        i += 1
            return False

        return kmp_search(serialize(root), serialize(subRoot))
```

Time Complexity: O(m + n) Space Complexity: O(m + n)

Similar Questions

- Count Univalue Subtrees
- Most Frequent Subtree Sum

Serialize and Deserialize Binary Tree

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

Clarification: The input/output format is the same as how LeetCode serializes a binary tree. You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Example 1:

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

Example 2:

Input: root = [] Output: []

Constraints:

```
The number of nodes in the tree is in the range [0, 104].
-1000 <= Node.val <= 1000
```

https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

Easy Solution

```
def easy_solution(root: Optional[TreeNode]) -> Optional[TreeNod
    # Use Codec class to serialize and deserialize
    codec = Codec()
    serialized = codec.serialize(root)
    return codec.deserialize(serialized)
```

Optimized Solution

```
def optimized_solution(root: Optional[TreeNode]) -> Optional[Tr
    # The easy solution is already optimal for this problem
    return easy_solution(root)
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Encode and Decode Strings

- Serialize and Deserialize BST
- Find Duplicate Subtrees
- Serialize and Deserialize N-ary Tree

Same Tree

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1: Input: p = [1,2,3], q = [1,2,3] Output: true

Example 2: Input: p = [1,2], q = [1,null,2] Output: false

Example 3: Input: p = [1,2,1], q = [1,1,2] Output: false

Constraints: The number of nodes in both trees is in the range [0, 100]. -104 <= Node.val <= 104

https://leetcode.com/problems/same-tree/description/

Easy Solution

```
def easy_solution(p: TreeNode, q: TreeNode) -> bool:
    # Recursive solution: Check each node
    if not p and not q:
        return True
    if not p or not q:
        return False
    return (p.val == q.val and
            easy_solution(p.left, q.left) and
            easy_solution(p.right, q.right))
```

Optimized Solution

```
def optimized_solution(p: TreeNode, q: TreeNode) -> bool:
    # Iterative solution: Use stack to compare nodes
    stack = [(p, q)]
    while stack:
        node1, node2 = stack.pop()
        if not node1 and not node2:
            continue
        if not node1 or not node2:
            return False
        if node1.val != node2.val:
            return False
        stack.append((node1.right, node2.right))
        stack.append((node1.left, node2.left))
    return True
```

Time Complexity: O(min(n, m)) Space Complexity: O(min(h1, h2))

Similar Questions

- Symmetric Tree
- Subtree of Another Tree

Maximum Depth of Binary Tree

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

Constraints: The number of nodes in the tree is in the range [0, 104]. -100 <= Node.val <= 100

https://leetcode.com/problems/maximum-depth-of-binary-tree/description/

Easy Solution

```
def easy_solution(root: TreeNode) -> int:
    # Recursive solution: Calculate depth for each subtree
    if not root:
        return 0
    return 1 + max(easy_solution(root.left), easy_solution(root
```

Optimized Solution

```
def optimized_solution(root: TreeNode) -> int:
    # Iterative solution: Use depth-first search with stack
    if not root:
        return 0

    stack = [(root, 1)]
    max_depth = 0

    while stack:
        node, depth = stack.pop()
        max_depth = max(max_depth, depth)

        if node.right:
            stack.append((node.right, depth + 1))
        if node.left:
            stack.append((node.left, depth + 1))

    return max_depth
```

Time Complexity: O(n) Space Complexity: O(h)

Similar Questions

- Balanced Binary Tree
- Minimum Depth of Binary Tree

Lowest Common Ancestor of a Binary Search Tree

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

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 = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8 Output: 6 Explanation: The LCA of nodes 2 and 8 is 6.

Example 2:

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4 Output: 2 Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

Example 3:

Input: root = [2,1], p = 2, q = 1 Output: 2

Constraints:

```
The number of nodes in the tree is in the range [2, 105].
-109 <= Node.val <= 109
All Node.val are unique.
p != q
p and q will exist in the BST.
```

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/

Easy Solution

```
def easy_solution(root: Optional[TreeNode], p: TreeNode, q: Tre
    # Recursive solution: Traverse the tree to find LCA
    if not root:
        return None
    if p.val < root.val and q.val < root.val:
        return easy_solution(root.left, p, q)
    if p.val > root.val and q.val > root.val:
        return easy_solution(root.right, p, q)
    return root
```

Optimized Solution

```
def optimized_solution(root: Optional[TreeNode], p: TreeNode, q
```

```
        # Iterative solution: Traverse the tree to find LCA
        current = root
        while current:
            if p.val < current.val and q.val < current.val:
                current = current.left
            elif p.val > current.val and q.val > current.val:
                current = current.right
            else:
                return current
```

Time Complexity: O(H) Space Complexity: O(1)

Similar Questions

- Lowest Common Ancestor of a Binary Tree
- Smallest Common Region

Kth Smallest Element in a BST

Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.

Example 1:

Input: root = [3,1,4,null,2], k = 1 Output: 1

Example 2:

Input: root = [5,3,6,2,4,null,null,1], k = 3 Output: 3

Constraints:

```
The number of nodes in the tree is n.
1 <= k <= n <= 104
0 <= Node.val <= 104
```

https://leetcode.com/problems/kth-smallest-element-in-a-bst/

Easy Solution

```
        def easy_solution(root: Optional[TreeNode], k: int) -> int:
            # Recursive solution: Inorder traversal to find kth smalles
            def inorder(node: Optional[TreeNode]) -> None:
                nonlocal k, result
                if not node:
                    return
                inorder(node.left)
                k -= 1
                if k == 0:
                    result = node.val
                    return
                inorder(node.right)

            result = None
            inorder(root)
            return result
```

Optimized Solution

```
        def optimized_solution(root: Optional[TreeNode], k: int) -> int
            # Iterative solution: Inorder traversal using stack
            stack = []
            current = root

            while current or stack:
                while current:
                    stack.append(current)
```

```
            current = current.left

        current = stack.pop()
        k -= 1
        if k == 0:
            return current.val

        current = current.right
```

Time Complexity: O(H + k) Space Complexity: O(H)

Similar Questions

- Binary Tree Inorder Traversal
- Second Minimum Node In a Binary Tree

Invert Binary Tree

Given the root of a binary tree, invert the tree, and return its root.

Example 1: Input: root = [4,2,7,1,3,6,9] Output: [4,7,2,9,6,3,1]

Example 2: Input: root = [2,1,3] Output: [2,3,1]

Example 3: Input: root = [] Output: []

Constraints: The number of nodes in the tree is in the range [0, 100]. -100 <= Node.val <= 100

https://leetcode.com/problems/invert-binary-tree/description/

Easy Solution

```
def easy_solution(root: TreeNode) -> TreeNode:
    # Recursive solution: Swap left and right subtrees
    if not root:
        return None
    root.left, root.right = easy_solution(root.right), easy_sol
    return root
```

Optimized Solution

```
def optimized_solution(root: TreeNode) -> TreeNode:
    # Iterative solution: Use queue to invert the tree
    if not root:
        return None

    queue = [root]
    while queue:
        node = queue.pop(0)
        node.left, node.right = node.right, node.left

        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)

    return root
```

Time Complexity: O(n) Space Complexity: O(h)

Similar Questions

- Reverse Odd Levels of Binary Tree

Implement Trie (Prefix Tree)

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 t
boolean startsWith(String prefix) Returns true if there is a previously
```

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 * 104 calls in total will be made to insert, search, and star
```

https://leetcode.com/problems/implement-trie-prefix-tree/

Easy Solution

```
def easy_solution():
    # Use the Trie class to perform operations
    return Trie()
```

Optimized Solution

```
def optimized_solution():
    # The easy solution is already optimal for this problem
    return Trie()
```

Time Complexity: O(m) Space Complexity: O(m)

Similar Questions

- Design Add and Search Words Data Structure
- Design Search Autocomplete System
- Replace Words

- Implement Magic Dictionary

Construct Binary Tree from Preorder and Inorder Traversal

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example 1:

Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7] Output: [3,9,20,null,null,15,7]

Example 2:

Input: preorder = [-1], inorder = [-1] Output: [-1]

Constraints:

```
1 <= preorder.length <= 3000
inorder.length == preorder.length
-3000 <= preorder[i], inorder[i] <= 3000
preorder and inorder consist of unique values.
Each value of inorder also appears in preorder.
preorder is guaranteed to be the preorder traversal of the tree.
inorder is guaranteed to be the inorder traversal of the tree.
```

https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

Easy Solution

```
def easy_solution(preorder: List[int], inorder: List[int]) -> O
    # Recursive solution: Use preorder and inorder to build the
    if not preorder or not inorder:
        return None
    root = TreeNode(preorder[0])
    mid = inorder.index(preorder[0])
    root.left = easy_solution(preorder[1:mid+1], inorder[:mid])
    root.right = easy_solution(preorder[mid+1:], inorder[mid+1:
    return root
```

Optimized Solution

```
def optimized_solution(preorder: List[int], inorder: List[int])
    # Optimized solution: Use a hashmap to quickly find the roo
    def build(start: int, end: int) -> Optional[TreeNode]:
        nonlocal pre_idx
        if start > end:
            return None
        root = TreeNode(preorder[pre_idx])
        pre_idx += 1
```

```
            mid = inorder_map[root.val]
            root.left = build(start, mid - 1)
            root.right = build(mid + 1, end)
            return root

        pre_idx = 0
        inorder_map = {val: idx for idx, val in enumerate(inorder)}
        return build(0, len(inorder) - 1)
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Construct Binary Tree from Inorder and Postorder Traversal

Binary Tree Maximum Path Sum

A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

Given the root of a binary tree, return the maximum path sum of any path.

Example 1:

Input: root = [1,2,3] Output: 6

Example 2:

Input: root = [-10,9,20,null,null,15,7] Output: 42

Constraints:

```
The number of nodes in the tree is in the range [1, 3 * 104].
-1000 <= Node.val <= 1000
```

https://leetcode.com/problems/binary-tree-maximum-path-sum/

Easy Solution

```
        def easy_solution(root: Optional[TreeNode]) -> int:
            # Helper function to calculate the maximum gain from each n
            def max_gain(node: Optional[TreeNode]) -> int:
                nonlocal max_sum
                if not node:
                    return 0
                left_gain = max(max_gain(node.left), 0)
                right_gain = max(max_gain(node.right), 0)
                path_sum = node.val + left_gain + right_gain
                max_sum = max(max_sum, path_sum)
                return node.val + max(left_gain, right_gain)

            max_sum = float('-inf')
            max_gain(root)
            return max_sum
```

Optimized Solution

```
        def optimized_solution(root: Optional[TreeNode]) -> int:
            # The easy solution is already optimal for this problem
            return easy_solution(root)
```

Time Complexity: O(n) Space Complexity: O(h)

Similar Questions

- Path Sum
- Sum Root to Leaf Numbers

Binary Tree Level Order Traversal

Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Example 1:

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

Example 2:

Input: root = [1] Output: [[1]]

Example 3:

Input: root = [] Output: []

Constraints:

```
The number of nodes in the tree is in the range [0, 2000].
-1000 <= Node.val <= 1000
```

https://leetcode.com/problems/binary-tree-level-order-traversal/

Easy Solution

```
        def easy_solution(root: Optional[TreeNode]) -> List[List[int]]:
            # Iterative solution: Use a queue to traverse each level
            if not root:
                return []
            result = []
            queue = [root]
            while queue:
                level_size = len(queue)
                level = []
                for _ in range(level_size):
                    node = queue.pop(0)
                    level.append(node.val)
                    if node.left:
                        queue.append(node.left)
                    if node.right:
                        queue.append(node.right)
                result.append(level)
            return result
```

Optimized Solution

```
        def optimized_solution(root: Optional[TreeNode]) -> List[List[i
            # More optimized iterative solution: Use a queue to travers
            if not root:
```

```
        return []
    result = []
    level = [root]
    while level:
        result.append([node.val for node in level])
        level = [child for node in level for child in (node.lef
    return result
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Binary Tree Zigzag Level Order Traversal
- Binary Tree Level Order Traversal II
- Minimum Depth of Binary Tree
- Binary Tree Vertical Order Traversal

Word Break

Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

Input: s = "leetcode", wordDict = ["leet","code"] Output: true Explanation: Return true because "leetcode" can be segmented as "leet code".

Example 2:

Input: s = "applepenapple", wordDict = ["apple","pen"] Output: true Explanation: Return true because "applepenapple" can be segmented as "apple pen apple". Note that you are allowed to reuse a dictionary word.

Example 3:

Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"] Output: false

Constraints:

```
1 <= s.length <= 300
1 <= wordDict.length <= 1000
1 <= wordDict[i].length <= 20
s and wordDict[i] consist of only lowercase English letters.
All the strings of wordDict are unique.
```

https://leetcode.com/problems/word-break/description/

Easy Solution

```
def easy_solution(s: str, wordDict: List[str]) -> bool:
    # Dynamic programming solution with O(n^2) time complexity
    dp = [False] * (len(s) + 1)
    dp[0] = True
    for i in range(1, len(s) + 1):
        for j in range(i):
            if dp[j] and s[j:i] in wordDict:
                dp[i] = True
                break
    return dp[len(s)]
```

Optimized Solution

```
def optimized_solution(s: str, wordDict: List[str]) -> bool:
    # Optimized dynamic programming solution using a set for wo
    word_set = set(wordDict)
```

```
        dp = [False] * (len(s) + 1)
        dp[0] = True
        for i in range(1, len(s) + 1):
            for j in range(i):
                if dp[j] and s[j:i] in word_set:
                    dp[i] = True
                    break
        return dp[len(s)]
```

Time Complexity: O(n^2) Space Complexity: O(n)

Similar Questions

- Word Break II

Unique Paths

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 * 109.

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
```

https://leetcode.com/problems/unique-paths/description/

Easy Solution

```
def easy_solution(m: int, n: int) -> int:
    # Dynamic programming solution with O(m * n) space complexi
    dp = [[1] * n for _ in range(m)]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    return dp[m-1][n-1]
```

Optimized Solution

```
def optimized_solution(m: int, n: int) -> int:
    # Optimized solution with O(n) space complexity
    row = [1] * n
    for _ in range(1, m):
        for j in range(1, n):
            row[j] += row[j-1]
    return row[-1]
```

Time Complexity: O(m * n) Space Complexity: O(n)

Similar Questions

- Unique Paths II
- Minimum Path Sum
- Dungeon Game

Longest Increasing Subsequence

Given an integer array nums, return the length of the longest strictly increasing subsequence.

Example 1:

Input: nums = [10,9,2,5,3,7,101,18] Output: 4 Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

Example 2:

Input: nums = [0,1,0,3,2,3] Output: 4

Example 3:

Input: nums = [7,7,7,7,7,7,7] Output: 1

Constraints:

```
1 <= nums.length <= 2500
-104 <= nums[i] <= 104
```

Follow up: Can you come up with an algorithm that runs in O(n log(n)) time complexity?

https://leetcode.com/problems/longest-increasing-subsequence/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> int:
    # Dynamic programming solution with O(n^2) time complexity
    if not nums:
        return 0
    dp = [1] * len(nums)
    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)
    return max(dp)
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> int:
    # Optimized solution using binary search with O(n log n) ti
    tails = [0] * len(nums)
    size = 0
    for num in nums:
        i, j = 0, size
        while i != j:
            m = (i + j) // 2
            if tails[m] < num:
```

```
                i = m + 1
            else:
                j = m
        tails[i] = num
        size = max(i + 1, size)
    return size
```

Time Complexity: O(n log n) Space Complexity: O(n)

Similar Questions

- Increasing Triplet Subsequence
- Russian Doll Envelopes
- Maximum Length of Pair Chain

Jump Game

You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return true if you can reach the last index, or false otherwise.

Example 1:

Input: nums = [2,3,1,1,4] Output: true Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [3,2,1,0,4] Output: false Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints:

```
1 <= nums.length <= 104
0 <= nums[i] <= 105
```

https://leetcode.com/problems/jump-game/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> bool:
    # Greedy solution with O(n) time complexity
    max_reach = 0
    for i, jump in enumerate(nums):
        if i > max_reach:
            return False
        max_reach = max(max_reach, i + jump)
    return True
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> bool:
    # Optimized greedy solution
    last_pos = len(nums) - 1
    for i in range(len(nums) - 1, -1, -1):
        if i + nums[i] >= last_pos:
            last_pos = i
    return last_pos == 0
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Jump Game II

House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system 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 = [2,3,2] Output: 3 Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.

Example 2:

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 3:

Input: nums = [1,2,3] Output: 3

Constraints:

```
1 <= nums.length <= 100
0 <= nums[i] <= 1000
```

https://leetcode.com/problems/house-robber-ii/description/

Easy Solution

```
    def easy_solution(nums: List[int]) -> int:
        return rob(nums)
```

Optimized Solution

```
    def optimized_solution(nums: List[int]) -> int:
        return rob(nums)
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- House Robber
- Paint House
- Paint Fence

- House Robber III
- Non-negative Integers without Consecutive Ones

House Robber

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
```

https://leetcode.com/problems/house-robber/description/

Easy Solution

```
def easy_solution(nums: List[int]) -> int:
    # Dynamic programming solution with O(n) space complexity
    if not nums:
        return 0
    if len(nums) <= 2:
        return max(nums)
    dp = [0] * len(nums)
    dp[0] = nums[0]
    dp[1] = max(nums[0], nums[1])
    for i in range(2, len(nums)):
        dp[i] = max(dp[i-1], dp[i-2] + nums[i])
    return dp[-1]
```

Optimized Solution

```
def optimized_solution(nums: List[int]) -> int:
    # Optimized solution with O(1) space complexity
    if not nums:
        return 0
    prev, curr = 0, 0
    for num in nums:
```

```
            prev, curr = curr, max(curr, prev + num)
        return curr
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Maximum Product Subarray
- House Robber II
- Paint House
- Paint Fence
- House Robber III

Decode Ways

A message containing letters from A-Z can be encoded into numbers using the following mapping:

'A' -> "1" 'B' -> "2" ... 'Z' -> "26"

To decode an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways).

For example, "11106" can be mapped into:

```
"AAJF" with the grouping (1 1 10 6)
"KJF" with the grouping (11 10 6)
```

Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".

Given a string s containing only digits, return the number of ways to decode it.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

Input: s = "12" Output: 2 Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: s = "226" Output: 3 Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Example 3:

Input: s = "06" Output: 0 Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06").

Constraints:

```
1 <= s.length <= 100
s contains only digits and may contain leading zero(s).
```

https://leetcode.com/problems/decode-ways/description/

Easy Solution

```
        def easy_solution(s: str) -> int:
            # Dynamic programming solution with O(n) space complexity
            if not s or s[0] == '0':
                return 0
            dp = [0] * (len(s) + 1)
            dp[0] = 1
            dp[1] = 1
            for i in range(2, len(s) + 1):
```

```
            if s[i-1] != '0':
                dp[i] += dp[i-1]
            if s[i-2] == '1' or (s[i-2] == '2' and s[i-1] <= '6'):
                dp[i] += dp[i-2]
        return dp[-1]
```

Optimized Solution

```
    def optimized_solution(s: str) -> int:
        # Optimized solution with O(1) space complexity
        if not s or s[0] == '0':
            return 0
        prev, curr = 1, 1
        for i in range(1, len(s)):
            temp = 0
            if s[i] != '0':
                temp = curr
            if s[i-1] == '1' or (s[i-1] == '2' and s[i] <= '6'):
                temp += prev
            prev, curr = curr, temp
        return curr
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Decode Ways II

Combination Sum

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

Input: candidates = [2,3,6,7], target = 7 Output: [[2,2,3],[7]] Explanation: 2 and 3 are the only numbers that can be combined to get 7. 2 can be used twice, and the combination [2,2,3] is valid. 7 can be used once, and the combination [7] is valid.

Example 2:

Input: candidates = [2,3,5], target = 8 Output: [[2,2,2,2],[2,3,3],[3,5]]

Example 3:

Input: candidates = [2], target = 1 Output: []

Constraints:

```
1 <= candidates.length <= 30
2 <= candidates[i] <= 40
All elements of candidates are distinct.
1 <= target <= 40
```

https://leetcode.com/problems/combination-sum/description/

Easy Solution

```
        def easy_solution(candidates: List[int], target: int) -> List[L
            # Backtracking solution
            def backtrack(start: int, target: int, path: List[int]) ->
                if target == 0:
                    result.append(path[:])
                    return
                for i in range(start, len(candidates)):
                    if candidates[i] > target:
                        break
                    path.append(candidates[i])
                    backtrack(i, target - candidates[i], path)
                    path.pop()

            result: List[List[int]] = []
            candidates.sort()
```

```
        backtrack(0, target, [])
        return result
```

Optimized Solution

```
    def optimized_solution(candidates: List[int], target: int) -> L
        # Dynamic programming solution
        dp: List[List[List[int]]] = [[] for _ in range(target + 1)]
        dp[0] = [[]]

        for c in candidates:
            for i in range(c, target + 1):
                dp[i].extend([comb + [c] for comb in dp[i - c]])

        return dp[target]
```

Time Complexity: O(n * target) Space Complexity: O(target)

Similar Questions

- Letter Combinations of a Phone Number
- Combination Sum II
- Combinations
- Combination Sum III
- Factor Combinations
- Combination Sum IV

Coin Change

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Example 1:

Input: coins = [1,2,5], amount = 11 Output: 3 Explanation: 11 = 5 + 5 + 1

Example 2:

Input: coins = [2], amount = 3 Output: -1

Example 3:

Input: coins = [1], amount = 0 Output: 0

Constraints:

```
1 <= coins.length <= 12
1 <= coins[i] <= 231 - 1
0 <= amount <= 104
```

https://leetcode.com/problems/coin-change/description/

Easy Solution

```
def easy_solution(coins: List[int], amount: int) -> int:
    # Dynamic programming solution with O(amount * len(coins))
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0
    for coin in coins:
        for x in range(coin, amount + 1):
            dp[x] = min(dp[x], dp[x - coin] + 1)
    return dp[amount] if dp[amount] != float('inf') else -1
```

Optimized Solution

```
def optimized_solution(coins: List[int], amount: int) -> int:
    # Optimized dynamic programming solution
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0
    for i in range(1, amount + 1):
        dp[i] = min(dp[i - c] if i - c >= 0 else float('inf') f
    return dp[amount] if dp[amount] != float('inf') else -1
```

Time Complexity: O(amount * len(coins)) Space Complexity: O(amount)

Similar Questions

- Minimum Cost For Tickets

Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2 Output: 2 Explanation: There are two ways to climb to the top.

1. 1 step + 1 step
2. 2 steps

Example 2:

Input: n = 3 Output: 3 Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

Constraints:

```
1 <= n <= 45
```

https://leetcode.com/problems/climbing-stairs/description/

Easy Solution

```
def easy_solution(n: int) -> int:
    # Dynamic programming solution with O(n) space complexity
    if n <= 2:
        return n
    dp = [0] * (n + 1)
    dp[1] = 1
    dp[2] = 2
    for i in range(3, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
```

Optimized Solution

```
def optimized_solution(n: int) -> int:
    # Optimized solution with O(1) space complexity
    if n <= 2:
        return n
    a, b = 1, 2
    for _ in range(3, n + 1):
        a, b = b, a + b
```

```
            return b
```

Time Complexity: O(n) Space Complexity: O(1)

Similar Questions

- Min Cost Climbing Stairs
- Fibonacci Number
- N-th Tribonacci Number

Pacific Atlantic Water Flow

There is an m x n rectangular island that borders both the Pacific Ocean and Atlantic Ocean. The Pacific Ocean touches the island's left and top edges, and the Atlantic Ocean touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an m x n integer matrix heights where heights[r][c] represents the height above sea level of the cell at coordinate (r, c).

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is less than or equal to the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return a 2D list of grid coordinates result where result[i] = [ri, ci] denotes that rain water can flow from cell (ri, ci) to both the Pacific and Atlantic oceans.

Example 1:

Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]] Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]] Explanation: The following cells can flow to the Pacific and Atlantic oceans, as shown below: [0,4]: [0,4] -> Pacific Ocean [0,4] -> Atlantic Ocean [1,3]: [1,3] -> Pacific Ocean [1,3] -> Atlantic Ocean [1,4]: [1,4] -> Pacific Ocean [1,4] -> Atlantic Ocean [2,2]: [2,2] -> Pacific Ocean [2,2] -> Atlantic Ocean [3,0]: [3,0] -> Pacific Ocean [3,0] -> Atlantic Ocean [3,1]: [3,1] -> Pacific Ocean [3,1] -> Atlantic Ocean [4,0]: [4,0] -> Pacific Ocean [4,0] -> Atlantic Ocean

Example 2:

Input: heights = [[1]] Output: [[0,0]] Explanation: The water can flow from the only cell to the Pacific and Atlantic oceans.

Constraints:

```
m == heights.length
n == heights[i].length
1 <= m, n <= 200
0 <= heights[i][j] <= 105
```

https://leetcode.com/problems/pacific-atlantic-water-flow/description/

Easy Solution

```
        def easy_solution(heights: List[List[int]]) -> List[List[int]]:
            # Easy solution: DFS from both oceans to find reachable cel
            if not heights or not heights[0]:
                return []

            m, n = len(heights), len(heights[0])
            pacific = set()
            atlantic = set()
```

```
        def dfs(i: int, j: int, reachable: set) -> None:
            reachable.add((i, j))
            for di, dj in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
                ni, nj = i + di, j + dj
                if 0 <= ni < m and 0 <= nj < n and (ni, nj) not in
                    dfs(ni, nj, reachable)

        for i in range(m):
            dfs(i, 0, pacific)
            dfs(i, n - 1, atlantic)

        for j in range(n):
            dfs(0, j, pacific)
            dfs(m - 1, j, atlantic)

        return list(pacific & atlantic)
```

Optimized Solution

```
    def optimized_solution(heights: List[List[int]]) -> List[List[i
        # Optimized solution: Use boolean arrays to track reachable
        if not heights or not heights[0]:
            return []

        m, n = len(heights), len(heights[0])
        pacific = [[False] * n for _ in range(m)]
        atlantic = [[False] * n for _ in range(m)]

        def dfs(i: int, j: int, reachable: List[List[bool]]) -> Non
            reachable[i][j] = True
            for di, dj in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
                ni, nj = i + di, j + dj
                if 0 <= ni < m and 0 <= nj < n and not reachable[ni
                    dfs(ni, nj, reachable)

        for i in range(m):
            dfs(i, 0, pacific)
            dfs(i, n - 1, atlantic)

        for j in range(n):
            dfs(0, j, pacific)
            dfs(m - 1, j, atlantic)

        return [[i, j] for i in range(m) for j in range(n) if pacif
```

Time Complexity: O(m * n) Space Complexity: O(m * n)

Similar Questions

- Number of Islands

- Surrounded Regions

Number of Islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input: grid = [["1","1","1","1","0"], ["1","1","0","1","0"], ["1","1","0","0","0"], ["0","0","0","0","0"]] Output: 1

Example 2:

Input: grid = [["1","0","0","0","0"], ["0","1","0","0","0"], ["0","0","1","0","0"], ["0","0","0","1","0"]] Output: 4

Constraints:

```
m == grid.length
n == grid[i].length
1 <= m, n <= 300
grid[i][j] is '0' or '1'.
```

https://leetcode.com/problems/number-of-islands/description/

Easy Solution

```
        def easy_solution(grid: List[List[str]]) -> int:
            # Easy solution: DFS to count islands
            if not grid or not grid[0]:
                return 0

            m, n = len(grid), len(grid[0])
            islands = 0

            def dfs(i: int, j: int) -> None:
                if i < 0 or i >= m or j < 0 or j >= n or grid[i][j] ==
                    return
                grid[i][j] = '0'
                dfs(i + 1, j)
                dfs(i - 1, j)
                dfs(i, j + 1)
                dfs(i, j - 1)

            for i in range(m):
                for j in range(n):
                    if grid[i][j] == '1':
                        islands += 1
                        dfs(i, j)
```

```
        return islands
```

Optimized Solution

```
    def optimized_solution(grid: List[List[str]]) -> int:
        # Optimized solution: BFS to count islands
        if not grid or not grid[0]:
            return 0

        m, n = len(grid), len(grid[0])
        islands = 0

        def bfs(i: int, j: int) -> None:
            queue = deque([(i, j)])
            while queue:
                i, j = queue.popleft()
                for di, dj in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
                    ni, nj = i + di, j + dj
                    if 0 <= ni < m and 0 <= nj < n and grid[ni][nj]
                        grid[ni][nj] = '0'
                        queue.append((ni, nj))

        for i in range(m):
            for j in range(n):
                if grid[i][j] == '1':
                    islands += 1
                    grid[i][j] = '0'
                    bfs(i, j)

        return islands
```

Time Complexity: O(m * n) Space Complexity: O(min(m, n))

Similar Questions

- Surrounded Regions
- Walls and Gates
- Number of Islands II
- Number of Distinct Islands
- Max Area of Island

Number of Connected Components in an Undirected Graph

You have a graph of n nodes. You are given an integer n and an array edges where edges[i] = [ai, bi] indicates that there is an edge between ai and bi in the graph.

Return the number of connected components in the graph.

Example 1:

Input: n = 5, edges = [[0,1],[1,2],[3,4]] Output: 2

Example 2:

Input: n = 5, edges = [[0,1],[1,2],[2,3],[3,4]] Output: 1

Constraints:

```
1 <= n <= 2000
0 <= edges.length <= 5000
edges[i].length == 2
0 <= ai, bi < n
ai != bi
There are no repeated edges.
```

https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/description/

Easy Solution

```
        def easy_solution(n: int, edges: List[List[int]]) -> int:
            # Easy solution: DFS to count connected components
            graph = [[] for _ in range(n)]
            for u, v in edges:
                graph[u].append(v)
                graph[v].append(u)

            visited = set()
            components = 0

            def dfs(node: int) -> None:
                visited.add(node)
                for neighbor in graph[node]:
                    if neighbor not in visited:
                        dfs(neighbor)

            for node in range(n):
                if node not in visited:
                    dfs(node)
                    components += 1
```

```
        return components
```

Optimized Solution

```
    def optimized_solution(n: int, edges: List[List[int]]) -> int:
        # Optimized solution: Union-Find to count connected compone
        parent = list(range(n))
        rank = [0] * n
        components = n

        def find(x: int) -> int:
            if parent[x] != x:
                parent[x] = find(parent[x])
            return parent[x]

        def union(x: int, y: int) -> None:
            nonlocal components
            root_x, root_y = find(x), find(y)
            if root_x != root_y:
                if rank[root_x] < rank[root_y]:
                    parent[root_x] = root_y
                elif rank[root_x] > rank[root_y]:
                    parent[root_y] = root_x
                else:
                    parent[root_y] = root_x
                    rank[root_x] += 1
                components -= 1

        for u, v in edges:
            union(u, v)

        return components
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Number of Islands
- Graph Valid Tree
- Friend Circles

Longest Consecutive Sequence

Given an unsorted array of integers nums, return the length of the longest consecutive elements sequence.

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

Example 1:

Input: nums = [100,4,200,1,3,2] Output: 4 Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

Example 2:

Input: nums = [0,3,7,2,5,8,4,6,0,1] Output: 9

Constraints:

```
0 <= nums.length <= 105
-109 <= nums[i] <= 109
```

https://leetcode.com/problems/longest-consecutive-sequence/description/

Easy Solution

```
        def easy_solution(nums: List[int]) -> int:
            # Easy solution: Use a set to find the longest consecutive
            if not nums:
                return 0

            num_set = set(nums)
            longest = 0

            for num in num_set:
                if num - 1 not in num_set:
                    current = num
                    streak = 1

                    while current + 1 in num_set:
                        current += 1
                        streak += 1

                    longest = max(longest, streak)

            return longest
```

Optimized Solution

```
        def optimized_solution(nums: List[int]) -> int:
            # Optimized solution: Similar approach but streamlined
```

```
        num_set = set(nums)
        longest = 0

        for num in nums:
            if num - 1 not in num_set:
                current = num
                streak = 1

                while current + 1 in num_set:
                    current += 1
                    streak += 1

                longest = max(longest, streak)

        return longest
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Binary Tree Longest Consecutive Sequence

Graph Valid Tree

You have a graph of n nodes labeled from 0 to n - 1. You are given an integer n and a list of edges where edges[i] = [ai, bi] indicates that there is an undirected edge between nodes ai and bi in the graph.

Return true if the edges of the given graph make up a valid tree, and false otherwise.

Example 1:

Input: n = 5, edges = [[0,1],[0,2],[0,3],[1,4]] Output: true

Example 2:

Input: n = 5, edges = [[0,1],[1,2],[2,3],[1,3],[1,4]] Output: false

Constraints:

```
1 <= n <= 2000
0 <= edges.length <= 5000
edges[i].length == 2
0 <= ai, bi < n
ai != bi
There are no self-loops or repeated edges.
```

https://leetcode.com/problems/graph-valid-tree/description/

Easy Solution

```
        def easy_solution(n: int, edges: List[List[int]]) -> bool:
            # Easy solution: DFS to check for cycles and connectivity
            if len(edges) != n - 1:
                return False

            graph = [[] for _ in range(n)]
            for u, v in edges:
                graph[u].append(v)
                graph[v].append(u)

            visited = set()

            def dfs(node: int, parent: int) -> bool:
                visited.add(node)
                for neighbor in graph[node]:
                    if neighbor == parent:
                        continue
                    if neighbor in visited:
                        return False
                    if not dfs(neighbor, node):
                        return False
                return True
```

```
        return dfs(0, -1) and len(visited) == n
```

Optimized Solution

```
    def optimized_solution(n: int, edges: List[List[int]]) -> bool:
        # Optimized solution: Union-Find to check for cycles and co
        if len(edges) != n - 1:
            return False

        parent = list(range(n))

        def find(x: int) -> int:
            if parent[x] != x:
                parent[x] = find(parent[x])
            return parent[x]

        def union(x: int, y: int) -> bool:
            root_x, root_y = find(x), find(y)
            if root_x == root_y:
                return False
            parent[root_x] = root_y
            return True

        return all(union(u, v) for u, v in edges)
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Number of Connected Components in an Undirected Graph
- Redundant Connection

Course Schedule

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

For example, the pair [0, 1], indicates that to take course 0 you have

Return true if you can finish all courses. Otherwise, return false.

Example 1:

Input: numCourses = 2, prerequisites = [[1,0]] Output: true Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input: numCourses = 2, prerequisites = [[1,0],[0,1]] Output: false Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0, and to take course 0 you should have finished course 1. So it is impossible.

Constraints:

```
1 <= numCourses <= 2000
0 <= prerequisites.length <= 5000
prerequisites[i].length == 2
0 <= ai, bi < numCourses
All the pairs prerequisites[i] are unique.
```

https://leetcode.com/problems/course-schedule/description/

Easy Solution

```
        def easy_solution(numCourses: int, prerequisites: List[List[int
            # Easy solution: DFS to detect cycles in the graph
            graph = [[] for _ in range(numCourses)]
            for course, prereq in prerequisites:
                graph[course].append(prereq)

            def has_cycle(course: int, path: set) -> bool:
                if course in path:
                    return True
                if not graph[course]:
                    return False
                path.add(course)
                for prereq in graph[course]:
                    if has_cycle(prereq, path):
                        return True
                path.remove(course)
                graph[course] = []
                return False
```

```
        for course in range(numCourses):
            if has_cycle(course, set()):
                return False
        return True
```

Optimized Solution

```
    def optimized_solution(numCourses: int, prerequisites: List[Lis
        # Optimized solution: Topological sort using Kahn's algorit
        graph = [[] for _ in range(numCourses)]
        in_degree = [0] * numCourses

        for course, prereq in prerequisites:
            graph[prereq].append(course)
            in_degree[course] += 1

        queue = deque([course for course in range(numCourses) if in
        taken = 0

        while queue:
            course = queue.popleft()
            taken += 1
            for next_course in graph[course]:
                in_degree[next_course] -= 1
                if in_degree[next_course] == 0:
                    queue.append(next_course)

        return taken == numCourses
```

Time Complexity: O(V + E) Space Complexity: O(V + E)

Similar Questions

- Course Schedule II
- Graph Valid Tree
- Minimum Height Trees
- Course Schedule III

Clone Graph

Given a reference of a node in a connected undirected graph.

Return a deep copy (clone) of the graph.

Each node in the graph contains a value (int) and a list (List[Node]) of its neighbors.

class Node { public int val; public List neighbors; }

Test case format:

For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with val == 1, the second node with val == 2, and so on. The graph is represented in the test case using an adjacency list.

An adjacency list is a collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with val = 1. You must return the copy of the given node as a reference to the cloned graph.

Example 1:

Input: adjList = [[2,4],[1,3],[2,4],[1,3]] Output: [[2,4],[1,3],[2,4],[1,3]] Explanation: There are 4 nodes in the graph. 1's neighbors are 2 and 4. 2's neighbors are 1 and 3. 3's neighbors are 2 and 4. 4's neighbors are 1 and 3.

Example 2:

Input: adjList = [[]] Output: [[]] Explanation: Note that the input contains one empty list. The graph consists of only one node with val = 1 since there are no neighbors.

Example 3:

Input: adjList = [] Output: [] Explanation: This an empty graph, it does not have any nodes.

Constraints:

```
The number of nodes in the graph is in the range [0, 100].
1 <= Node.val <= 100
Node.val is unique for each node.
There are no repeated edges and no self-loops in the graph.
The Graph is connected and all nodes can be visited starting from the g
```

https://leetcode.com/problems/clone-graph/description/

Easy Solution

```
        def easy_solution(node: Optional[Node]) -> Optional[Node]:
            # Easy solution: DFS with recursion to clone the graph
            if not node:
                return None
```

```
    visited = {}

    def dfs(node: Node) -> Node:
        if node in visited:
            return visited[node]

        clone = Node(node.val)
        visited[node] = clone

        for neighbor in node.neighbors:
            clone.neighbors.append(dfs(neighbor))

        return clone

    return dfs(node)
```

Optimized Solution

```
def optimized_solution(node: Optional[Node]) -> Optional[Node]:
    # Optimized solution: BFS with queue to clone the graph
    if not node:
        return None

    visited = {}
    queue = deque([node])
    visited[node] = Node(node.val)

    while queue:
        current = queue.popleft()
        for neighbor in current.neighbors:
            if neighbor not in visited:
                visited[neighbor] = Node(neighbor.val)
                queue.append(neighbor)
            visited[current].neighbors.append(visited[neighbor]

    return visited[node]
```

Time Complexity: O(N + E) Space Complexity: O(N)

Similar Questions

- Copy List with Random Pointer
- Clone Binary Tree With Random Pointer
- Clone N-ary Tree

Alien Dictionary

There is a new alien language that uses the English alphabet. However, the order among letters are unknown to you.

You are given a list of strings words from the alien language's dictionary, where the strings in words are sorted lexicographically by the rules of this new language.

Return a string of the unique letters in the new alien language sorted in lexicographically increasing order by the new language's rules. If there is no solution, return "". If there are multiple solutions, return any of them.

A string s is lexicographically smaller than a string t if at the first letter where they differ, the letter in s comes before the letter in t in the alien language. If the first min(s.length, t.length) letters are the same, then s is smaller if and only if s.length < t.length.

Example 1:

Input: words = ["wrt","wrf","er","ett","rftt"] Output: "wertf"

Example 2:

Input: words = ["z","x"] Output: "zx"

Example 3:

Input: words = ["z","x","z"] Output: "" Explanation: The order is invalid, so return "".

Constraints:

```
1 <= words.length <= 100
1 <= words[i].length <= 100
words[i] consists of only lowercase English letters.
```

https://leetcode.com/problems/alien-dictionary/description/

Easy Solution

```
        def easy_solution(words: List[str]) -> str:
            # Easy solution: DFS to build the order
            graph: Dict[str, set] = {c: set() for word in words for c i

            for i in range(len(words) - 1):
                w1, w2 = words[i], words[i + 1]
                min_len = min(len(w1), len(w2))
                if len(w1) > len(w2) and w1[:min_len] == w2[:min_len]:
                    return ""
                for j in range(min_len):
                    if w1[j] != w2[j]:
                        graph[w1[j]].add(w2[j])
                        break
```

```
    visited = {}
    result = []

    def dfs(c: str) -> bool:
        if c in visited:
            return visited[c]
        visited[c] = True
        for nei in graph[c]:
            if dfs(nei):
                return True
        visited[c] = False
        result.append(c)
        return False

    for c in graph:
        if dfs(c):
            return ""

    return "".join(result[::-1])
```

Optimized Solution

```
def optimized_solution(words: List[str]) -> str:
    # Optimized solution: BFS with in-degree tracking
    graph: Dict[str, set] = {c: set() for word in words for c i
    in_degree: Dict[str, int] = {c: 0 for word in words for c i

    for i in range(len(words) - 1):
        w1, w2 = words[i], words[i + 1]
        min_len = min(len(w1), len(w2))
        if len(w1) > len(w2) and w1[:min_len] == w2[:min_len]:
            return ""
        for j in range(min_len):
            if w1[j] != w2[j]:
                if w2[j] not in graph[w1[j]]:
                    graph[w1[j]].add(w2[j])
                    in_degree[w2[j]] += 1
                break

    queue = deque([c for c in in_degree if in_degree[c] == 0])
    result = []

    while queue:
        c = queue.popleft()
        result.append(c)
        for nei in graph[c]:
            in_degree[nei] -= 1
            if in_degree[nei] == 0:
                queue.append(nei)

    if len(result) != len(graph):
```

```
            return ""
        return "".join(result)
```

Time Complexity: O(C) Space Complexity: O(1)

Similar Questions

- Course Schedule II
- Sequence Reconstruction

Non-overlapping Intervals

Given an array of intervals intervals where intervals[i] = [starti, endi], return the minimum number of intervals you need to remove to make the rest of the intervals 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 <= intervals.length <= 105
intervals[i].length == 2
-5 * 104 <= starti < endi <= 5 * 104
```

https://leetcode.com/problems/non-overlapping-intervals/description/

Easy Solution

```
        def easy_solution(intervals: List[List[int]]) -> int:
            # Easy solution: Sort by end time and count overlaps
            if not intervals:
                return 0

            intervals.sort(key=lambda x: x[1])
            count = 0
            end = float('-inf')

            for interval in intervals:
                if interval[0] >= end:
                    end = interval[1]
                else:
                    count += 1

            return count
```

Optimized Solution

```
        def optimized_solution(intervals: List[List[int]]) -> int:
```

```
        # Optimized solution: Similar to easy solution but starts w
        if not intervals:
            return 0

        intervals.sort(key=lambda x: x[1])
        count = 0
        end = intervals[0][1]

        for i in range(1, len(intervals)):
            if intervals[i][0] < end:
                count += 1
            else:
                end = intervals[i][1]

        return count
```

Time Complexity: O(n log n) Space Complexity: O(1)

Similar Questions

- Minimum Number of Arrows to Burst Balloons

Merge Intervals

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:

Input: intervals = [[1,3],[2,6],[8,10],[15,18]] Output: [[1,6],[8,10],[15,18]] Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].

Example 2:

Input: intervals = [[1,4],[4,5]] Output: [[1,5]] Explanation: Intervals [1,4] and [4,5] are considered overlapping.

Constraints:

```
1 <= intervals.length <= 104
intervals[i].length == 2
0 <= starti <= endi <= 104
```

https://leetcode.com/problems/merge-intervals/description/

Easy Solution

```
        def easy_solution(intervals: List[List[int]]) -> List[List[int]
            # Easy solution: Sort intervals and merge sequentially
            if not intervals:
                return []

            intervals.sort(key=lambda x: x[0])
            merged = [intervals[0]]

            for interval in intervals[1:]:
                if interval[0] <= merged[-1][1]:
                    merged[-1][1] = max(merged[-1][1], interval[1])
                else:
                    merged.append(interval)

            return merged
```

Optimized Solution

```
        def optimized_solution(intervals: List[List[int]]) -> List[List
            # Optimized solution: Similar to easy solution but checks i
            if not intervals:
                return []

            intervals.sort(key=lambda x: x[0])
```

```
        merged = []

        for interval in intervals:
            if not merged or merged[-1][1] < interval[0]:
                merged.append(interval)
            else:
                merged[-1][1] = max(merged[-1][1], interval[1])

        return merged
```

Time Complexity: O(n log n) Space Complexity: O(n)

Similar Questions

- Insert Interval
- Meeting Rooms
- Meeting Rooms II
- Teemo Attacking
- Add Bold Tag in String
- Range Module
- Employee Free Time
- Partition Labels

Meeting Rooms II

Given an array of meeting time intervals intervals where intervals[i] = [starti, endi], return the minimum number of conference rooms required.

Example 1:

Input: intervals = [[0,30],[5,10],[15,20]] Output: 2

Example 2:

Input: intervals = [[7,10],[2,4]] Output: 1

Constraints:

```
0 <= intervals.length <= 104
intervals[i].length == 2
0 <= starti < endi <= 106
```

https://leetcode.com/problems/meeting-rooms-ii/description/

Easy Solution

```
def easy_solution(intervals: List[List[int]]) -> int:
    # Easy solution: Sort start and end times and use two point
    start_times = sorted(interval[0] for interval in intervals)
    end_times = sorted(interval[1] for interval in intervals)

    rooms = 0
    end_ptr = 0

    for start in start_times:
        if start < end_times[end_ptr]:
            rooms += 1
        else:
            end_ptr += 1

    return rooms
```

Optimized Solution

```
def optimized_solution(intervals: List[List[int]]) -> int:
    # Optimized solution: Use a list of events and sort them
    events = []
    for start, end in intervals:
        events.append((start, 1))
        events.append((end, -1))

    events.sort(key=lambda x: (x[0], -x[1]))
```

```
    rooms = 0
    max_rooms = 0

    for _, event_type in events:
        rooms += event_type
        max_rooms = max(max_rooms, rooms)

    return max_rooms
```

Time Complexity: O(n log n) Space Complexity: O(n)

Similar Questions

- Merge Intervals
- Meeting Rooms
- Minimum Number of Arrows to Burst Balloons
- Car Pooling

Meeting Rooms

Given an array of meeting time intervals where intervals[i] = [starti, endi], determine if a person could attend all meetings.

Example 1:

Input: intervals = [[0,30],[5,10],[15,20]] Output: false

Example 2:

Input: intervals = [[7,10],[2,4]] Output: true

Constraints:

```
0 <= intervals.length <= 104
intervals[i].length == 2
0 <= starti < endi <= 106
```

https://leetcode.com/problems/meeting-rooms/description/

Easy Solution

```
def easy_solution(intervals: List[List[int]]) -> bool:
    # Easy solution: Sort intervals by start time and check for
    intervals.sort(key=lambda x: x[0])

    for i in range(1, len(intervals)):
        if intervals[i][0] < intervals[i-1][1]:
            return False

    return True
```

Optimized Solution

```
def optimized_solution(intervals: List[List[int]]) -> bool:
    # Optimized solution: Sort start and end times separately a
    start_times = sorted(interval[0] for interval in intervals)
    end_times = sorted(interval[1] for interval in intervals)

    for i in range(1, len(intervals)):
        if start_times[i] < end_times[i-1]:
            return False

    return True
```

Time Complexity: O(n log n) Space Complexity: O(n)

Similar Questions

- Merge Intervals
- Meeting Rooms II

Insert Interval

You are given an array of non-overlapping intervals intervals where intervals[i] = [starti, endi] represent the start and the end of the ith interval and intervals is sorted in ascending order by starti. You are also given an interval newInterval = [start, end] that represents the start and end of another interval.

Insert newInterval into intervals such that intervals is still sorted in ascending order by starti and intervals still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return intervals after the insertion.

Example 1:

Input: intervals = [[1,3],[6,9]], newInterval = [2,5] Output: [[1,5],[6,9]]

Example 2:

Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8] Output: [[1,2], [3,10],[12,16]] Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

Constraints:

```
0 <= intervals.length <= 104
intervals[i].length == 2
0 <= starti <= endi <= 105
intervals is sorted by starti in ascending order.
newInterval.length == 2
0 <= start <= end <= 105
```

https://leetcode.com/problems/insert-interval/description/

Easy Solution

```
        def easy_solution(intervals: List[List[int]], newInterval: List
            # Easy solution: Iterate through intervals, merge if overla
            result = []
            i = 0
            n = len(intervals)

            # Add intervals before newInterval
            while i < n and intervals[i][1] < newInterval[0]:
                result.append(intervals[i])
                i += 1

            # Merge overlapping intervals
            while i < n and intervals[i][0] <= newInterval[1]:
                newInterval[0] = min(newInterval[0], intervals[i][0])
                newInterval[1] = max(newInterval[1], intervals[i][1])
                i += 1
```

```
        result.append(newInterval)

        # Add remaining intervals
        while i < n:
            result.append(intervals[i])
            i += 1

        return result
```

Optimized Solution

```
    def optimized_solution(intervals: List[List[int]], newInterval:
        # Optimized solution: Iterate through intervals and merge o
        result = []
        for interval in intervals:
            if interval[1] < newInterval[0]:
                result.append(interval)
            elif interval[0] > newInterval[1]:
                result.append(newInterval)
                newInterval = interval
            else:
                newInterval[0] = min(newInterval[0], interval[0])
                newInterval[1] = max(newInterval[1], interval[1])
        result.append(newInterval)
        return result
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Merge Intervals
- Range Module

Word Search

Given an m x n grid of characters board and a string word, return true if word exists in the grid.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:

Input: board = [['A','B','C','E'],['S','F','C','S'],['A','D','E','E']], word = "ABCCED" Output: true

Example 2:

Input: board = [['A','B','C','E'],['S','F','C','S'],['A','D','E','E']], word = "SEE" Output: true

Example 3:

Input: board = [['A','B','C','E'],['S','F','C','S'],['A','D','E','E']], word = "ABCB" Output: false

Constraints:

```
m == board.length
n == board[i].length
1 <= m, n <= 6
1 <= word.length <= 15
board and word consists of only lowercase and uppercase English letters
```

https://leetcode.com/problems/word-search/description/

Easy Solution

```
        def easy_solution(board: List[List[str]], word: str) -> bool:
            # Brute-force solution: Use DFS to check for the word
            def dfs(i: int, j: int, k: int) -> bool:
                if k == len(word):
                    return True
                if (i < 0 or i >= len(board) or
                    j < 0 or j >= len(board[0]) or
                    board[i][j] != word[k]):
                    return False

                temp, board[i][j] = board[i][j], '#'
                result = (dfs(i+1, j, k+1) or
                          dfs(i-1, j, k+1) or
                          dfs(i, j+1, k+1) or
                          dfs(i, j-1, k+1))
                board[i][j] = temp
                return result

            for i in range(len(board)):
```

```
            for j in range(len(board[0])):
                if dfs(i, j, 0):
                    return True
        return False
```

Optimized Solution

```
    def optimized_solution(board: List[List[str]], word: str) -> bo
        # Optimized solution: Use DFS with visited array
        def dfs(i: int, j: int, k: int) -> bool:
            if k == len(word):
                return True
            if (i < 0 or i >= m or j < 0 or j >= n or
                visited[i][j] or board[i][j] != word[k]):
                return False

            visited[i][j] = True
            for di, dj in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
                if dfs(i + di, j + dj, k + 1):
                    return True
            visited[i][j] = False
            return False

        m, n = len(board), len(board[0])
        visited = [[False] * n for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if board[i][j] == word[0] and dfs(i, j, 0):
                    return True
        return False
```

Time Complexity: O(m * n * 4^L) Space Complexity: O(m * n)

Similar Questions

- Word Search II

Spiral Matrix

Given an m x n matrix, return all elements of the matrix in spiral order.

Example 1:

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]] Output: [1,2,3,6,9,8,7,4,5]

Example 2:

Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]] Output: [1,2,3,4,8,12,11,10,9,5,6,7]

Constraints:

```
m == matrix.length
n == matrix[0].length
1 <= m, n <= 10
-100 <= matrix[i][j] <= 100
```

https://leetcode.com/problems/spiral-matrix/description/

Easy Solution

```
        def easy_solution(matrix: List[List[int]]) -> List[int]:
            # Brute-force solution: Traverse the matrix in spiral order
            if not matrix:
                return []

            result = []
            top, bottom, left, right = 0, len(matrix) - 1, 0, len(matri

            while top <= bottom and left <= right:
                for j in range(left, right + 1):
                    result.append(matrix[top][j])
                top += 1

                for i in range(top, bottom + 1):
                    result.append(matrix[i][right])
                right -= 1

                if top <= bottom:
                    for j in range(right, left - 1, -1):
                        result.append(matrix[bottom][j])
                    bottom -= 1

                if left <= right:
                    for i in range(bottom, top - 1, -1):
                        result.append(matrix[i][left])
                    left += 1

            return result
```

Optimized Solution

```
        def optimized_solution(matrix: List[List[int]]) -> List[int]:
            # Optimized solution: Use direction vectors to traverse the
            if not matrix:
                return []

            result = []
            m, n = len(matrix), len(matrix[0])
            directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
            d = 0
            row, col = 0, 0
            visited = set()

            for _ in range(m * n):
                result.append(matrix[row][col])
                visited.add((row, col))

                next_row, next_col = row + directions[d][0], col + dire

                if (next_row < 0 or next_row >= m or
                    next_col < 0 or next_col >= n or
                    (next_row, next_col) in visited):
                    d = (d + 1) % 4
                    next_row, next_col = row + directions[d][0], col +

                row, col = next_row, next_col

            return result
```

Time Complexity: O(m * n) Space Complexity: O(1)

Similar Questions

- Spiral Matrix II
- Spiral Matrix III

Set Matrix Zeroes

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

You must do it in place.

Example 1:

Input: matrix = [[1,1,1],[1,0,1],[1,1,1]] Output: [[1,0,1],[0,0,0],[1,0,1]]

Example 2:

Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]] Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]

Constraints:

```
m == matrix.length
n == matrix[0].length
1 <= m, n <= 200
-231 <= matrix[i][j] <= 231 - 1
```

https://leetcode.com/problems/set-matrix-zeroes/description/

Easy Solution

```
        def easy_solution(matrix: List[List[int]]) -> None:
            # Brute-force solution: Use sets to track zero rows and col
            m, n = len(matrix), len(matrix[0])
            zero_rows, zero_cols = set(), set()

            for i in range(m):
                for j in range(n):
                    if matrix[i][j] == 0:
                        zero_rows.add(i)
                        zero_cols.add(j)

            for i in range(m):
                for j in range(n):
                    if i in zero_rows or j in zero_cols:
                        matrix[i][j] = 0
```

Optimized Solution

```
        def optimized_solution(matrix: List[List[int]]) -> None:
            # Optimized solution: Use first row and column as markers
            m, n = len(matrix), len(matrix[0])
            first_row_zero = False
            first_col_zero = False

            # Check if the first row has any zeros
```

```
        for j in range(n):
            if matrix[0][j] == 0:
                first_row_zero = True
                break

        # Check if the first column has any zeros
        for i in range(m):
            if matrix[i][0] == 0:
                first_col_zero = True
                break

        # Use first row and column to mark zeros
        for i in range(1, m):
            for j in range(1, n):
                if matrix[i][j] == 0:
                    matrix[i][0] = 0
                    matrix[0][j] = 0

        # Set matrix elements to zero based on markers
        for i in range(1, m):
            for j in range(1, n):
                if matrix[i][0] == 0 or matrix[0][j] == 0:
                    matrix[i][j] = 0

        # Set the first row to zero if needed
        if first_row_zero:
            for j in range(n):
                matrix[0][j] = 0

        # Set the first column to zero if needed
        if first_col_zero:
            for i in range(m):
                matrix[i][0] = 0
```

Time Complexity: O(m * n) Space Complexity: O(1)

Similar Questions

- Game of Life

Rotate Image

You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Example 1:

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]] Output: [[7,4,1],[8,5,2],[9,6,3]]

Example 2:

Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]] Output: [[15,13,2,5], [14,3,4,1],[12,6,8,9],[16,7,10,11]]

Constraints:

```
n == matrix.length == matrix[i].length
1 <= n <= 20
-1000 <= matrix[i][j] <= 1000
```

https://leetcode.com/problems/rotate-image/description/

Easy Solution

```
def easy_solution(matrix: List[List[int]]) -> None:
    # Brute-force solution: Transpose the matrix, then reverse
    n = len(matrix)

    # Transpose the matrix
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i

    # Reverse each row
    for i in range(n):
        matrix[i].reverse()
```

Optimized Solution

```
def optimized_solution(matrix: List[List[int]]) -> None:
    # Optimized solution: Rotate four rectangles
    n = len(matrix)

    # Rotate four rectangles
    for i in range(n // 2 + n % 2):
        for j in range(n // 2):
            tmp = matrix[n - 1 - j][i]
```

```
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1]
                matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 - i]
                matrix[j][n - 1 - i] = matrix[i][j]
                matrix[i][j] = tmp
```

Time Complexity: O(n^2) Space Complexity: O(1)

Similar Questions

- Determine Whether Matrix Can Be Obtained By Rotation

Top K Frequent Elements

Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.

Example 1:

Input: nums = [1,1,1,2,2,3], k = 2 Output: [1,2]

Example 2:

Input: nums = [1], k = 1 Output: [1]

Constraints:

```
1 <= nums.length <= 105
-104 <= nums[i] <= 104
k is in the range [1, the number of unique elements in the array].
It is guaranteed that the answer is unique.
```

https://leetcode.com/problems/top-k-frequent-elements/description/

Easy Solution

```
def easy_solution(nums: List[int], k: int) -> List[int]:
    # Brute-force solution: Use a heap to find the k most frequ
    count = {}
    for num in nums:
        count[num] = count.get(num, 0) + 1

    return heapq.nlargest(k, count.keys(), key=count.get)
```

Optimized Solution

```
def optimized_solution(nums: List[int], k: int) -> List[int]:
    # Optimized solution: Use bucket sort to find the k most fr
    count = {}
    freq = [[] for _ in range(len(nums) + 1)]

    for num in nums:
        count[num] = count.get(num, 0) + 1
    for num, c in count.items():
        freq[c].append(num)

    res = []
    for i in range(len(freq) - 1, 0, -1):
        for num in freq[i]:
            res.append(num)
            if len(res) == k:
                return res
```

Time Complexity: O(n) Space Complexity: O(n)

Similar Questions

- Top K Frequent Words
- Sort Characters By Frequency

Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]] Output: [1,1,2,3,4,4,5,6] Explanation: The linked-lists are: [1->4->5, 1->3->4, 2->6] merging them into one sorted list: 1->1->2->3->4->4->5->6

Example 2:

Input: lists = [] Output: []

Example 3:

Input: lists = [[]] Output: []

Constraints:

```
k == lists.length
0 <= k <= 104
0 <= lists[i].length <= 500
-104 <= lists[i][j] <= 104
lists[i] is sorted in ascending order.
The sum of lists[i].length will not exceed 104.
```

https://leetcode.com/problems/merge-k-sorted-lists/description/

Easy Solution

```
        def easy_solution(lists: List[ListNode]) -> ListNode:
            # Brute-force solution: Collect all nodes and sort them
            nodes = []
            head = point = ListNode(0)
            for l in lists:
                while l:
                    nodes.append(l.val)
                    l = l.next
            for x in sorted(nodes):
                point.next = ListNode(x)
                point = point.next
            return head.next
```

Optimized Solution

```
        def optimized_solution(lists: List[ListNode]) -> ListNode:
            # Optimized solution: Use a heap to merge lists
```

```
        heap = []
        for i, l in enumerate(lists):
            if l:
                heapq.heappush(heap, (l.val, i, l))

        dummy = ListNode(0)
        curr = dummy
        while heap:
            val, i, node = heapq.heappop(heap)
            curr.next = ListNode(val)
            curr = curr.next
            if node.next:
                heapq.heappush(heap, (node.next.val, i, node.next))

        return dummy.next
```

Time Complexity: O(N log k) Space Complexity: O(k)

Similar Questions

- Merge Two Sorted Lists
- Ugly Number II

Find Median from Data Stream

The MedianFinder class has two methods:

void addNum(int num) - Adds the integer num from the data stream to the data structure. double findMedian() - Returns the median of all elements so far. Answers within 10-5 of the actual answer will be accepted.

Implement the MedianFinder class: MedianFinder() initializes the MedianFinder object. void addNum(int num) adds the integer num from the data stream to the data structure. double findMedian() returns the median of all elements so far. Answers within 10-5 of the actual answer will be accepted.

Example 1:

Input ["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"] [[], [1], [2], [], [3], []] Output [null, null, null, 1.5, null, 2.0]

Explanation MedianFinder medianFinder = new MedianFinder(); medianFinder.addNum(1); // arr = [1] medianFinder.addNum(2); // arr = [1, 2] medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2) medianFinder.addNum(3); // arr[1, 2, 3] medianFinder.findMedian(); // return 2.0

Constraints:

```
-105 <= num <= 105
There will be at least one element in the data structure before calling
At most 5 * 104 calls will be made to addNum and findMedian.
```

https://leetcode.com/problems/find-median-from-data-stream/description/

Easy Solution

```
        def easy_solution() -> MedianFinder:
            return MedianFinder()
```

Optimized Solution

```
        def optimized_solution() -> MedianFinder:
            return MedianFinder()
```

Time Complexity: O(log n) for addNum, O(1) for findMedian Space Complexity: O(n)

Similar Questions

- Sliding Window Median
- Find Median from Data Stream II