0 - Exercises list:

lunes, 6 de mayo de 2024 8:31

https://leetcode.com/explore/interview/card/top-interview-questions-easy

Arrays: Remove duplicates

sábado, 4 de mayo de 2024 9:58

https://leetcode.com/explore/interview/card/top-interview-questions-easy/92/array/727/

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in* nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).

```python
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        # Case nums is empty.
        if not nums:
            return 0

        unique_ptr = 0

        for i in range(1, len(nums)):
            if nums[i]!=nums[unique_ptr]:
                # Move the pointer for unique element.
                unique_ptr += 1
                # Copy unique element to its rightful position.
                nums[unique_ptr] = nums[i]

        return unique_ptr + 1
```

Arrays: Best time to buy and sell stock

sábado, 4 de mayo de 2024 9:59

You are given an integer array prices where prices[i] is the price of a given stock on the i^{th} day.
On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.
Find and return *the **maximum** profit you can achieve*.

Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.

```python
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if not prices or len(prices)==1:
            return 0

        max_profit = 0
        for i in range(1, len(prices)):
            if prices[i] > prices[i-1]:
                max_profit += prices[i] - prices[i-1]

        return max_profit
```

Arrays: Rotate the array by k steps

sábado, 4 de mayo de 2024 11:19

Given an integer array nums, rotate the array to the right by k steps, where k is non-negative.

```
# Calculate the actual rotation steps needed (if k=7 and len=5 its the same as k=
2)
# because after every 5 steps, the array will be back to its original position.
```

```
class Solution(object):
    def rotate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: None Do not return anything, modify nums in-place instead.
        """
        if not nums or len(nums)==1:
            return

        k = k % len(nums) # Rotation steps needed.

        # Reversing the entire array.
        nums.reverse()

        # Reverse the first k elements.
        nums[:k] = reversed(nums[:k])

        # Reverse the remaining elements.
        nums[k:] = reversed(nums[k:])
```

Arrays: Check if has duplicate

sábado, 4 de mayo de 2024 11:46

```python
class Solution(object):
    def containsDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        map = {}
        for i in nums:
            if i not in map:
                map[i] = 'seen'
            else:
                return True
        return False
```

Efficient in time complexity O(n)

```python
class Solution(object):
    def containsDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        if not nums or len(nums)==1:
            return False

        nums.sort()

        for i in range(1, len(nums)):
            if nums[i] == nums[i-1]:
                return True

        return False
```

Efficient in memory usage.
But time complexity is sort O(n log n) + O(n)

Arrays: All nums appear twice but one

sábado, 4 de mayo de 2024 12:18

Find that one:

```python
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums)==1:
            return nums[0]

        result = 0
        for num in nums:
            result ^= num # XOR operation if num appears twice = 0.

        return result
```

^= performs a XOR operation. 1 1 = 0 , 0, 0 = 0, 0 1 = 1 , 1 0 = 1.

Arrays: Intersection of two arrays

sábado, 4 de mayo de 2024 12:54

Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

```python
class Solution(object):

    def efficient_intersection(self, short_nums, long_nums):

        intersection = []
        for l_num in long_nums:
            if l_num in short_nums:
                intersection.append(l_num)
                short_nums.remove(l_num)
        return intersection

    def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        if len(nums1) > len(nums2):
            return self.efficient_intersection(nums2, nums1)
        else:
            return self.efficient_intersection(nums1, nums2)
```

```python
class Solution(object):

    def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """

        freq_map = {}

        for num in nums1:
            freq_map[num] = freq_map.get(num, 0) + 1 # Gets value from key 'num' if not
                                                     # there return default value 0.
        intersect = []
        for num in nums2:
            if num in freq_map and freq_map[num] > 0:
                intersect.append(num)
                freq_map[num] -= 1 # Decrease the freq.

        return intersect
```

Let's analyze both approaches:

Using a Hashmap:

- This approach iterates through both arrays to create a frequency map for one array and then iterates through the other array to check for intersection. Creating the frequency map takes O(n)O(n)time, and iterating through the second array takes O(m)O(m)time, where nnand mmare the lengths of the arrays respectively. Overall time complexity is O(n+m)O(n+m).
- This approach doesn't modify the input arrays and doesn't rely on removing elements, which could potentially incur additional overhead.

Using Removal from a Copy:

- This approach iterates through one of the arrays and checks for each element if it exists in the other array, then removes it if found. Removing an element from a list using remove() operation takes O(n)O(n)time, where nnis the length of the list.
- The time complexity of this approach depends on the length of the longer array, as it iterates through that array. In the worst-case scenario where there are no common elements, it could take O(n·m)O(n·m)time, where nnand mmare the lengths of the arrays. Additionally, remove() operation may need to shift elements in the list, which adds extra overhead.
- This approach modifies the input arrays by removing elements, which may not be desirable if you want to keep the original arrays intact.
- This version has better space complexity since no additional storage elements are used (apart from the result list).

Arrays: Plus one

sábado, 4 de mayo de 2024 13:05

You are given a **large integer** represented as an integer array digits, where each digits[i] is the i^{th} digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.
Increment the large integer by one and return *the resulting array of digits*.

```
class Solution(object):
    def plusOne(self, digits):
        """
        :type digits: List[int]
        :rtype: List[int]
        """
        str_int = ''.join(map(str,digits))
        str_int = str(int(str_int)+1)

        return [int(str_digit) for str_digit in str_int]

```

Arrays: Move Zeroes

sábado, 4 de mayo de 2024 16:52

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.
Note that you must do this in-place without making a copy of the array.

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

```python
def movezeroes(nums):
    """
    :type nums: List[int]
    :rtype: int
    """
    slow_ptr = fast_ptr = 0

    while fast_ptr < len(nums):
        # Swap when non zero is found.
        if nums[fast_ptr] != 0 and fast_ptr != slow_ptr: # Avoids self-swap.
            nums[fast_ptr], nums[slow_ptr] = nums[slow_ptr], nums[fast_ptr]
        # Updates slow pointer.
        if nums[slow_ptr] != 0:
            slow_ptr = slow_ptr + 1
        fast_ptr += 1

    return nums
```

Swap_ptr points to the position for the swap. Pointer increases if a swap is performed or if its not placed in a position with a 0 value. (fast and slow pointers increase together until a position with value 0 is found)
If there is no need for a position swap, meaning we are in nums[fast_ptr] == 0, just increase the fast_ptr till a non zero pos is found.

Arrays: Two Sum

sábado, 4 de mayo de 2024 17:37

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].

Space complexity efficient solution:

```
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """

        for i in range(len(nums)):
            for j in range(len(nums)):
                if i!=j and nums[i] + nums[j] == target:
                    return [i, j]

        return []
```

Time complexity efficient solution:

```
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        hash_map = {}
        for i, val in enumerate(nums):
            # Case value has been found.
            if val in hash_map:
                return [i, hash_map[val]]

            hash_map[target - val] = i # Value we seek in the list.


        return []
```

Arrays: Check if sudoku is valid

sábado, 4 de mayo de 2024 18:56

```python
class Solution(object):

    def checkgrid(self, grid_check, row_idx, col_idx, element):

        for col_idx_range, col_grid in grid_check.items():
            if col_idx in col_idx_range:
                for row_idx_range, row_grid_dict in col_grid.items():
                    if row_idx in row_idx_range:
                        # Duplicate in grid.
                        if element in row_grid_dict:
                            return False
                        row_grid_dict[element] = None
                        # Terminate loops when corresponding dict is found.
                        break
                break
        return True


    def isValidSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: bool
        """

        col_check = {i: {} for i in range(10)}
        grid_check = {
                      (0, 1, 2): {(0, 1, 2): {},
                                  (3, 4, 5): {},
                                  (6, 7, 8): {}},
                      (3, 4, 5): {(0, 1, 2): {},
                                  (3, 4, 5): {},
                                  (6, 7, 8): {}},
                      (6, 7, 8): {(0, 1, 2): {},
                                  (3, 4, 5): {},
                                  (6, 7, 8): {}}
                     }
        for row_idx, row in enumerate(board):
            row_dict = {}
            for col_idx, element in enumerate(row):
                # Ignore empty values.
                if element != '.':
                    # If digit not 1-9.
                    if int(element) < 1  or int(element) > 9:
                        return False
                    # If duplicate in row.
                    if element in row_dict:
                        return False
                    # If duplicate in col:
                    if element in col_check[col_idx]:
                        return False
                    # If duplicate in grid:
                    if not self.checkgrid(grid_check, row_idx, col_idx, element):
                        return False

                    row_dict[element] = None
                    col_check[col_idx][element] = None

        return True
```

More efficient:

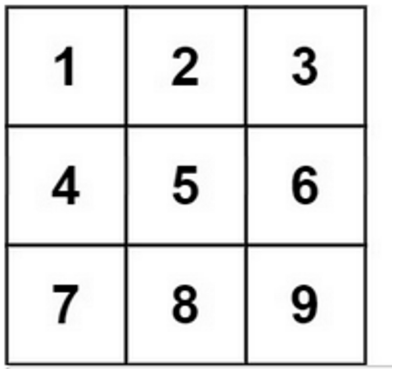
```python
class Solution(object):

    def isValidSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: bool
        """

        col_check = {i:{} for i in range(9)}
        box_check = {i:{} for i in range(9)}

        for row_idx, row in enumerate(board):
            row_check = {}
            for col_idx, element in enumerate(row):
                if element == '.':
                    continue # Goes to the next iteration.
                # Row duplicate check:
                if element in row_check:
                    return False
                # Col duplicate check:
                if element in col_check[col_idx]:
                    return False
                # Box duplicate check:
                box_idx = (row_idx // 3) * 3 + (col_idx // 3)
                if element in box_check[box_idx]:
                    return False

                row_check[element] = None
                col_check[col_idx][element] = None
                box_check[box_idx][element] = None

        return True
```

Arrays: Rotate image

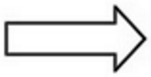
sábado, 4 de mayo de 2024 19:38

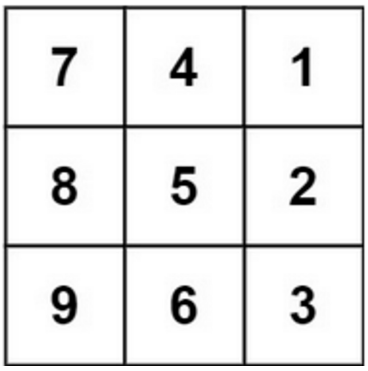
```python
class Solution(object):
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: None Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix)

        # Swap elements along the diagonal
        for i in range(n):
            for j in range(i+1, n):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

        # Reverse each row horizontally
        for i in range(n):
            matrix[i] = matrix[i][::-1]

        return matrix
```

1. Mirror the elements across the diagonal.
2. Reverse the rows.



-> [1 4 7]
[2 5 8]
[3 6 9]





Strings : Reverse integer

domingo, 5 de mayo de 2024 9:51

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

```
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        x_str_list = list(str(abs(x)))
        x_str_list.reverse()

        result = int(''.join(x_str_list))
        if result > (2 ** 31) - 1:
            return 0

        if x < 0:
            return -result
        else:
            return result
```

Strings: Reverse string.

domingo, 5 de mayo de 2024 9:55

```
class Solution(object):
    def reverseString(self, s):
        """
        :type s: List[str]
        :rtype: None Do not return anything, modify s in-place instead.
        """
        for i in range(len(s)//2):
            s[i], s[-1-i] = s[-1-i], s[i]
```

Where len(s) // 2 is the number of swaps needed.

A quicker solution would be to simply.

S.reverse()

Strings: First unique char in s

domingo, 5 de mayo de 2024 10:30

Given a string s, *find the first non-repeating character in it and return its index*. If it does not exist, return -1.

```python
from collections import OrderedDict


class Solution(object):
    def firstUniqChar(self, s):
        """
        :type s: str
        :rtype: int
        """

        result = OrderedDict()

        for i, val in enumerate(list(s)):
            if val in result:
                result[val][1] += 1
            else:
                result[val] = [i, 1]

        for key, value in result.items():
            if value[1] == 1:
                return value[0]

        return -1
```

With py3 all dicts are ordered dicts. (maintain order of insertion)

Strings: Anagram

domingo, 5 de mayo de 2024 10:52

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.

```
class Solution(object):
    def isAnagram(self, s, t):
        """
        :type s: str
        :type t: str
        :rtype: bool
        """

        anagram = {}

        for char in s:
            anagram[char] = anagram.get(char, 0)+1

        for char in t:
            if not char in anagram:
                return False

            anagram[char] -= 1

        for val in anagram.values():
            if val != 0:
                return False

        return True
```

Strings: Valid palindrome

domingo, 5 de mayo de 2024 10:53

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*.

```
class Solution(object):
    def isPalindrome(self, s):
        """
        :type s: str
        :rtype: bool
        """
        if s == '':
            return True

        s = ''.join(char for char in s if char.isalnum())
        s = s.lower()

        return s == s[-1::-1]
```

Linked Lists: Del node

domingo, 5 de mayo de 2024 15:17

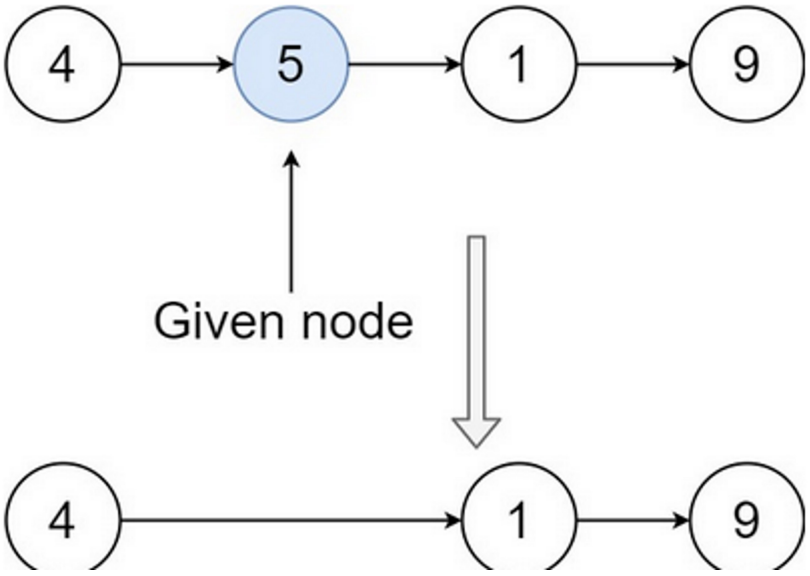
There is a singly-linked list head and we want to delete a node node in it.
You are given the node to be deleted node. You will **not be given access** to the first node of head.
All the values of the linked list are **unique**, and it is guaranteed that the given node node is not the last node in the linked list.
Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:

- The value of the given node should not exist in the linked list.
- The number of nodes in the linked list should decrease by one.
- All the values before node should be in the same order.
- All the values after node should be in the same order.

Custom testing:

- For the input, you should provide the entire linked list head and the node to be given node. node should not be the last node of the list and should be an actual node in the list.
- We will build the linked list and pass the node to your function.
- The output will be the entire list after calling your function.

Example 1:



```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        # Copy next val to current node and update pointer.
        node.val = node.next.val
        node.next = node.next.next
```

Linked Lists: Remove Nth Node From End of List

domingo, 5 de mayo de 2024 21:20

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        # Remove first element for list size 1.
        if head.next is None:
            return None

        # Used when element to supress is the first one.
        dummy_node = ListNode(next=head)

        fast_ptr = dummy_node
        slow_ptr = dummy_node

        c = 0
        # When the loop stops, our slow_ptr will point to the node to remove.
        while fast_ptr.next:
            fast_ptr = fast_ptr.next

            if c >= n:
                slow_ptr = slow_ptr.next
            c+=1

        # Element to remove at the begining.
        if slow_ptr == dummy_node:
            return slow_ptr.next.next # Head of new list.

        # Element to remove at the middle.
        if slow_ptr.next.next:
            slow_ptr.next.val = slow_ptr.next.next.val
            slow_ptr.next = slow_ptr.next.next
            return dummy_node.next

        # Element to remove at the end.
        slow_ptr.next = None

        return dummy_node.next # Ingore dummy node, return head.

```

Linked Lists: Reverse list

lunes, 6 de mayo de 2024 8:30

```python
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """

        itr = head
        prev_node = None

        # Case list size 1.
        if itr is None or itr.next is None:
            return head

        while itr.next:
            tmp = itr.next
            # Changing itr.next to point to prev node.
            itr.next = prev_node
            prev_node = itr
            # Moving the itr to the next node.
            itr = tmp

        # Change for the last node.
        itr.next = prev_node

        return itr # new head is last item from list.
```

Linked Lists: Merge two sorted lists

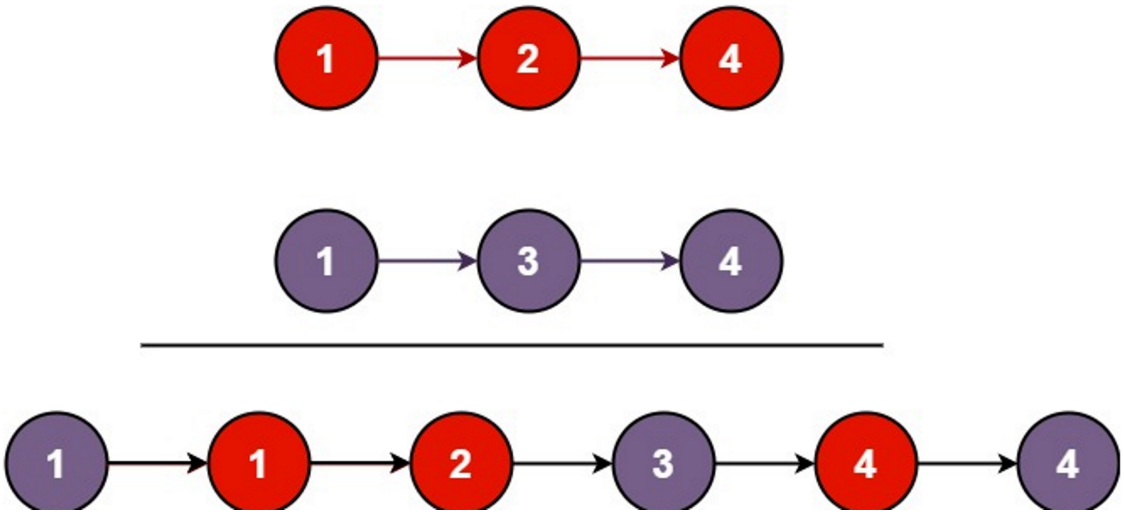
lunes, 6 de mayo de 2024 11:38

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

Merge the two lists into 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:



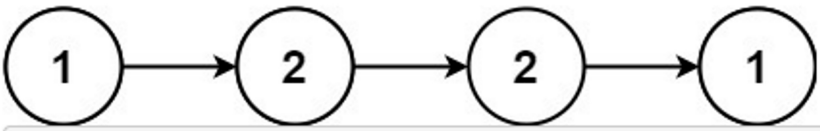
```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def mergeTwoLists(self, list1, list2):
        """
        :type list1: Optional[ListNode]
        :type list2: Optional[ListNode]
        :rtype: Optional[ListNode]
        """

        # Case any list is empty.
        if not list1:
            return list2

        if not list2:
            return list1

        dummy = ListNode() # Head of the merged list.
        merged_tail = dummy

        itr1 = list1
        itr2 = list2

        while itr1 and itr2:
            if itr1.val <= itr2.val:
                merged_tail.next = itr1
                itr1 = itr1.next
            else:
                merged_tail.next = itr2
                itr2 = itr2.next

            merged_tail = merged_tail.next

        # Add remaining elements to the merged list.
        if itr1:
            merged_tail.next = itr1
        elif itr2:
            merged_tail.next = itr2

        return dummy.next
```

Linked Lists: Palindrome

lunes, 6 de mayo de 2024 12:31

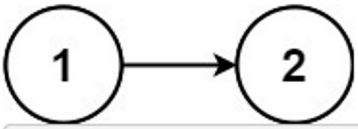
Given the `head` of a singly linked list, return `true` *if it is a palindrome or* `false` *otherwise.*

Example 1:




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

Example 2:




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

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def isPalindrome(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if not head or head.next is None:
            return True

        fast_itr = slow_itr = head

        # Find the middle: O(n/2) complexity.
        while fast_itr and fast_itr.next:
            slow_itr = slow_itr.next
            fast_itr = fast_itr.next.next


        # Reverse second half: O(n/2) complexity.
        prev = None
        while slow_itr:
            tmp = slow_itr.next
            slow_itr.next = prev
            prev = slow_itr
            slow_itr = tmp

        # Check if palindrome: O(n/2) complexity.
        first_half = head
        second_half = prev

        while second_half: # They are separated lists after the reversal
                           # first_half contains middle value in case of odd len.
            if first_half.val != second_half.val:
                return False
            first_half = first_half.next
            second_half = second_half.next

        # Total complexity = time = n + n/2 -> O(n) the fastest growing term.

        return True
```

Linked Lists: Cycle

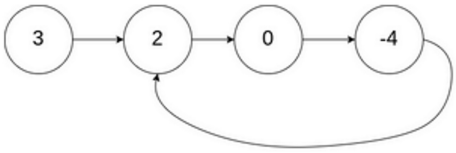
lunes, 6 de mayo de 2024 12:59

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.**

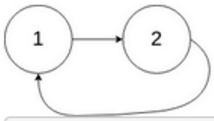
Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

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:




```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        visited = {}

        itr = head
        while itr not in visited:
            if not itr or not itr.next:
                return False

            visited[itr] = ''
            itr = itr.next

        if itr in visited:
            return True

        return False
```

This has better time complexity since O(n) but we are using more memory.

A solution where execution time slightly increases (we are doing one loop and a half aprox), still O(n)
but has a constant space complexity:

```python
class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        # Case no list is given or list has size 1.
        if not head or not head.next:
            return False

        slow_itr = head
        fast_itr = head.next # So we can enter the while.
        while fast_itr != slow_itr:
            # Case list has no cycle:
            if not fast_itr or not fast_itr.next:
                return False

            fast_itr = fast_itr.next.next
            slow_itr = slow_itr.next

        # If there is a cycle, before reaching n^2 fast_itr would be equal to slow_itr.
        return True
```

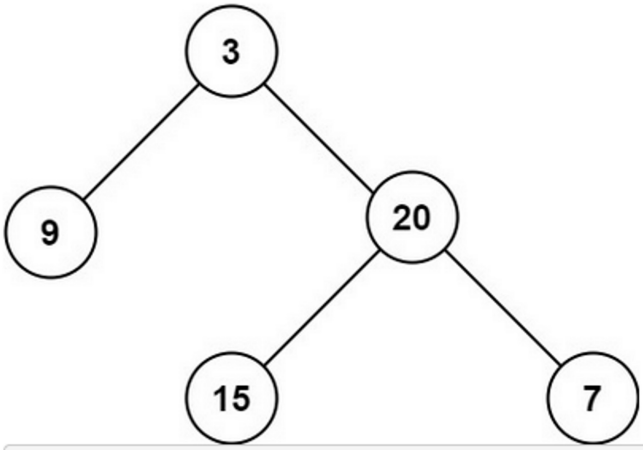
Trees: Maximum Depth of Binary Tree

lunes, 6 de mayo de 2024 17:47

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
```

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):

    def maxDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """

        if not root:
            return 0

        left_depth = self.maxDepth(root.left)
        right_depth = self.maxDepth(root.right)

        return max(left_depth, right_depth) + 1  # +1 for the root node
```

Trees: is BST?

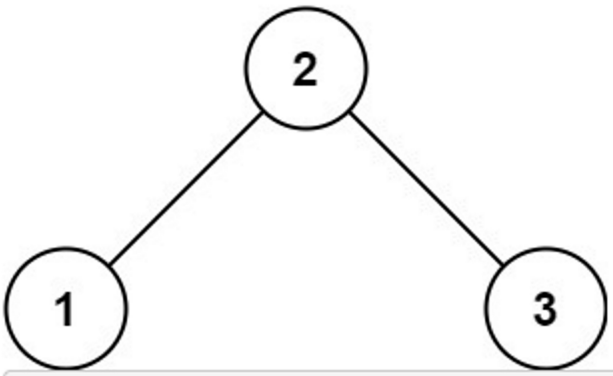
lunes, 6 de mayo de 2024 19:20

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
```

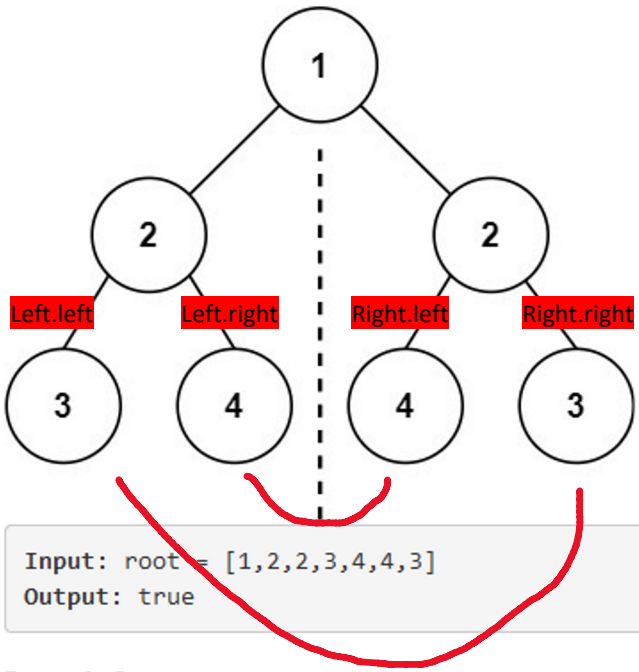
```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def isValidBST(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        def in_order_traversal(node, checker):
            if node:
                # Traverse left.
                in_order_traversal(node.left, checker)

                checker.append(node.val)

                in_order_traversal(node.right, checker)

        if not root or not root.left and not root.right:
            return True

        # List to store the values from the tree in order.
        checker = []
        # Fills the list from least significant val to most significant.
        # It does that by checking first the most left part of the tree.
        in_order_traversal(root, checker)

        for i in range(len(checker)-1):
            if checker[i] >= checker[i+1]:
                return False

        return True
```

Trees: is Symmetric?

martes, 7 de mayo de 2024 9:47

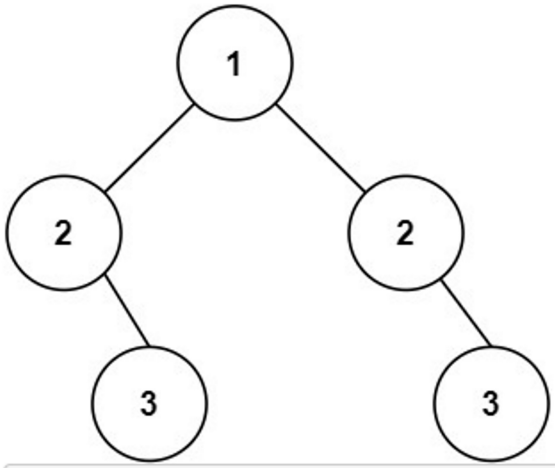
Given the `root` of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

Example 1:




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

Example 2:




```
Input: root = [1,2,2,null,3,null,3]
Output: false
```

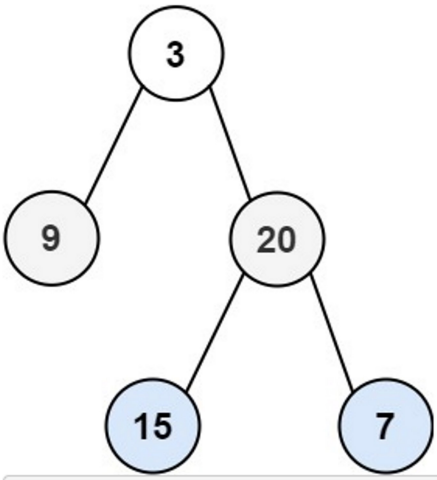
```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def isSymmetric(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        def is_mirror(left, right):
            if not left and not right:
                return True

            if not left or not right or left.val != right.val:
                return False

            return is_mirror(left.left, right.right) and is_mirror(left.right, right.left)

        return is_mirror(root.left, root.right) if root else True
```

Trees: Tree Level Order Traversal

martes, 7 de mayo de 2024 11:07

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]]
```

Iteratively solution:

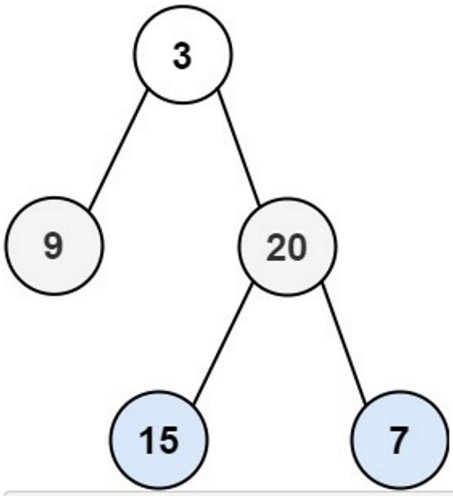
```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []

        result = []
        queue = [root]

        while queue:
            level_result = []
            for _ in range(len(queue)): # Loop for lvl size.
                # Dequeuing the first element.
                node = queue.pop(0)
                level_result.append(node.val)

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

            result.append(level_result)

        return result
```

Trees: LvL order traverse

martes, 7 de mayo de 2024 11:58

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]]
```

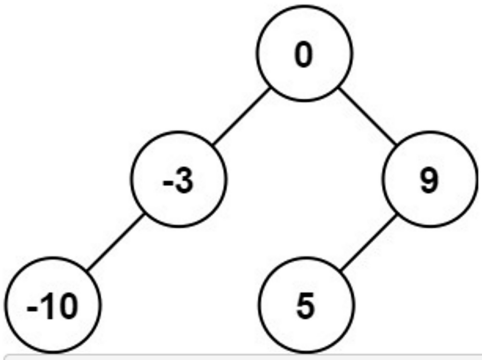
```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        if not root:
            return []

        result = []

        def lvl_traverse(node, lvl):
            # Create lvl if it does not exist.
            if len(result) == lvl:
                result.append([])

            result[lvl].append(node.val)

            if node.left:
                lvl_traverse(node.left, lvl + 1)

            if node.right:
                lvl_traverse(node.right, lvl + 1)

        lvl_traverse(root, 0)
        return result
```

Trees: Convert Sorted Array to BST

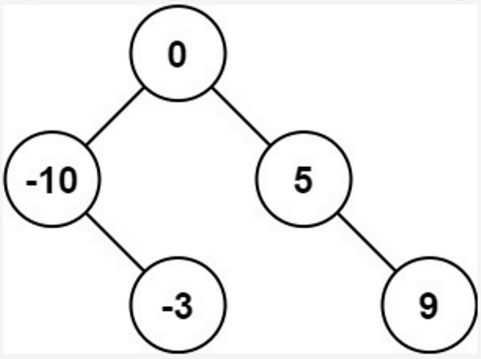
martes, 7 de mayo de 2024 12:57

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

Example 1:




```
Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```




```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def sortedArrayToBST(self, nums):
        """
        :type nums: List[int]
        :rtype: TreeNode
        """
        if not nums:
            return

        mid_idx = len(nums) // 2
        root = TreeNode(val=nums[mid_idx])

        root.left = self.sortedArrayToBST(nums[:mid_idx])
        root.right = self.sortedArrayToBST(nums[mid_idx+1:])

        return root
```

Sorting and Searching: Merge Sorted Array.

martes, 7 de mayo de 2024 14:01

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to `0` and should be ignored. `nums2` has a length of `n`.

Example 1:

```
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.
```

```python
class Solution(object):
    def merge(self, nums1, m, nums2, n):
        """
        :type nums1: List[int]
        :type m: int
        :type nums2: List[int]
        :type n: int
        :rtype: None Do not return anything, modify nums1 in-place instead.
        """

        if not nums1 or (m==0 and len(nums1) > 0):
            nums1[:] = nums2[:]

        bkws_itr = len(nums1) - 1
        n1_itr = m - 1
        n2_itr = n - 1
        while bkws_itr >= 0 and n2_itr >= 0 and n1_itr >= 0:
            if nums1[n1_itr] > nums2[n2_itr]:
                nums1[bkws_itr] = nums1[n1_itr]
                n1_itr -= 1
            elif nums1[n1_itr] == nums2[n2_itr]:
                nums1[bkws_itr] = nums1[n1_itr]
                bkws_itr -= 1
                n1_itr -= 1
                nums1[bkws_itr] = nums2[n2_itr]
                n2_itr -= 1
            else:
                nums1[bkws_itr] = nums2[n2_itr]
                n2_itr -= 1
            bkws_itr -= 1

        # Case n > m:
        if n1_itr <= 0 and n2_itr >= 0:
            while bkws_itr >= 0:
                nums1[bkws_itr] = nums2[n2_itr]
                bkws_itr -= 1
                n2_itr -= 1
```

Complexity O(m+n)