Neetcode Blind 75

1. Array and Hashing
2. Contains Duplicate

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

def containsDuplicate(nums):

    hashSet = set()

    for num in nums:

        if num in hashSet:

            return True

        else:

            hashSet.add(num)

    return False

arr = [1,2,3]

print(containsDuplicate(arr))

o/p

False

1. Valid Anagram

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

from collections import Counter

#Solution number 1, Time = O(n), Space = O(n)

def isAnagram1(s,t):

    if len(s) != len(t):

        return False

    hashMapS = {}

    hashMapT = {}

    for i in range(len(s)):

        hashMapS[s[i]] = 1 + hashMapS.get(s[i], 0)

        hashMapT[t[i]] = 1 + hashMapT.get(t[i], 0)

    for key in hashMapS:

        if hashMapS[key] != hashMapT.get(key, 0):

            return False

    return True

#Solution number 2, Time = O(nlogn), Space = O(n)

def isAnagram2(s,t):

    return sorted(s) == sorted(t)

#Solution number 3

def isAnagram3(s,t):

    return Counter(s) == Counter(t)

s = "anagram"

t = "nagaram"

print(isAnagram1(s, t))

print(isAnagram1(s, t))

print(isAnagram1(s, t))

o/p

False

False

False

1. Two Sum

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

def twoSum(nums, target):

    counter = {}

    possibilities = []

    for i in range(len(nums)):

        diff = target - nums[i]

        if nums[i] in counter:

            possibilities.append((counter[nums[i]], i))

        else:

            counter[diff] = i

    return possibilities

nums = [1,2,5,6,4,7]

target = 7

print(twoSum(nums, target))

o/p

[(1, 2), (0, 3)]

1. Group Anagrams

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

from collections import defaultdict

def groupAnagrams(strs):

    result = defaultdict(list)

    for s in strs:

        count = [0] \* 26

        for c in s:

            count[ord(c) - ord("a")] += 1

        result[tuple(count)].append(s)

    return result.values()

strs = ["bat", "tab", "rat", "tar", "eat", "ate", "tea"]

res = groupAnagrams(strs)

print(res)

o/p

dict\_values([['bat', 'tab'], ['rat', 'tar'], ['eat', 'ate', 'tea']])

1. Top K frequent Elements

<https://leetcode.com/problems/top-k-frequent-elements/>

def topKElements(nums, k):

    record = [[] for \_ in range(len(nums) + 1)]

    counter = {}

    for num in nums:

        counter[num] = 1 + counter.get(num, 0)

    for key, value in counter.items():

        record[value].append(key)

    result = []

    for i in range(len(record) - 1, -1, -1):

        if len(result) == k:

            break

        if record[i] == []:

            continue

        else:

            result.extend(record[i])

    return result

arr =[1,1,1,2,2,3,4,4,4,4,5,5,5,5]

k = 3

print(topKElements(arr, k))

o/p

[4, 5, 1]

1. Product of an Array except self

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

def productExceptSelf(nums):

    res = [1] \* len(nums)

    prefix = 1

    for i in range(len(nums)):

        res[i] = prefix

        prefix = prefix \* nums[i]

    postfix = 1

    for i in range(len(nums) - 1, -1, -1):

        res[i] = res[i] \* postfix

        postfix = postfix \* nums[i]

    return res

nums = [1,2,3,4,5,6]

result = productExceptSelf(nums)

print(result)

o/p

[720, 360, 240, 180, 144, 120]

1. Valid Sudoku

<https://leetcode.com/problems/valid-sudoku/description/>

def isValidSudoku(board):

    def isSudokuValid(r, c, num):

        for i in range(9):

            if board[r][i] == num and i != c or board[i][c] == num and i != r:

                return False

        quadr = r//3

        quadc = c//3

        for i in range(3\*quadr,3\*quadr+3):

            for j in range(3\*quadc,3\*quadc+3):

                if i!=r and j!=c and board[i][j] == num:

                    return False

        return True

    ROWS = len(board)

    COLS = len(board[0])

    for r in range(ROWS):

        for c in range(COLS):

            if board[r][c] !='.' and not isSudokuValid(r,c,board[r][c]):

                return False

    return True

board = [

    [5,3,'.','.',7,'.','.','.','.'],

    [6,'.','.',1,9,5,'.','.','.'],

    ['.',9,8,'.','.','.','.',6,'.'],

    [8,'.','.','.',6,'.','.','.',3],

    [4,'.','.',8,'.',3,'.','.',1],

    [7,3,'.','.',7,'.','.','.',6],

    ['.',6,'.','.','.','.',2,8,'.'],

    ['.','.','.',4,1,9,'.','.',5],

    ['.','.','.','.',8,'.','.',7,9],

]

res = isValidSudoku(board)

print(res)

o/p

False

1. Longest Consecutive sequence

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

def longConsecutiveSeq(nums):

    def checkSeq(n):

        seqLength = 0

        while n + seqLength in s:

            seqLength += 1

        return seqLength

    longest = 0

    s = set(nums)

    for num in nums:

        if num - 1 in s:

            continue

        longest = max(longest, checkSeq(num))

    return longest

nums = [100,3,1,2,200,4,10,11,12,13,14,15,16]

res = longConsecutiveSeq(nums)

print(res)

o/p

7

1. Two Pointers
2. Valid Palindrome

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

def validPalindrome(s):

    def checkChar(c):

        return ord(c) >= ord("a") and ord(c) <= ord("z") or \

        ord(c) >= ord("0") and ord(c) <= ord("9") or \

        ord(c) >= ord("A") and ord(c) <= ord("Z")

    i,j = 0, len(s) - 1

    while i < j:

        while i < j and not checkChar(s[i]):

            i += 1

        while j > i and not checkChar(s[j]):

            j -= 1

        if s[i].lower() == s[j].lower():

            i,j = i + 1, j - 1

        else:

            return False

    return True

s = 'A man, a plan, a canal panama'

print(validPalindrome(s))

o/p

True

1. Two Sum II – Input Array is sorted

<https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/description/>

def twoSumInSortedArray(arr, target):

    i,j = 0, len(arr) - 1

    while i < j:

        if (arr[i] + arr[j]) < target:

            i += 1

        elif (arr[i] + arr[j]) > target:

            j -= 1

        else:

            return i+1,j+1

    return []

arr = [1,2,3,4,5,6]

target = 8

print(twoSumInSortedArray(arr, target))

o/p

(2, 6)

1. 3 Sum

<https://leetcode.com/problems/3sum/>

def threeSum(nums):

    nums.sort()

    result = []

    for i in range(0,len(nums)):

        if i > 0 and nums[i] == nums[i - 1]:

            continue

        j,k = i + 1, len(nums) - 1

        while j < k:

            sum = nums[i] + nums[j] + nums[k]

            if sum == 0:

                result.append([nums[i],nums[j],nums[k]])

                j += 1

                while nums[j - 1] == nums[j] and j < k:

                    j += 1

            elif sum > 0:

                k -= 1

            else:

                j += 1

    return result

arr = [1,4,5,-5,-7,0,-7,-1,2,20,-5]

[-5, -7, -7, 2, -1, 0, 1, 2, 4, 3, 20]

print(threeSum(arr))

o/p

[[-7, 2, 5], [-5, 0, 5], [-5, 1, 4], [-1, 0, 1]]

1. Container with Most Water

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

def mostWater(heights):

    l,r = 0,len(heights) - 1

    mostWater =  float("-inf")

    while l < r:

        water = min(heights[l], heights[r]) \* (r - l)

        mostWater = max(water, mostWater)

        if heights[l] > heights[r]:

            r -= 1

        else:

            l += 1

    return mostWater

    ...

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

res = mostWater(heights)

print(res)

o/p

49

1. Trapping Rain Water

<https://leetcode.com/problems/trapping-rain-water/>

def trappedRainWater(height):

    l,r = 0,len(height) - 1

    maxLeft = height[l]

    maxRight = height[r]

    waterTrapped = 0

    while l < r:

        if maxLeft < maxRight:

            l += 1

            maxLeft = max(height[l], maxLeft)

            waterTrapped += maxLeft - height[l]

        else:

            r -= 1

            maxRight = max(height[r], maxRight)

            waterTrapped += maxRight - height[r]

    return waterTrapped

height = [1,5,3,6,6,1,5,4,3,6,4,8,5]

print(trappedRainWater(height))

o/p

15

1. Stacks
2. Evaluate Reverse Polish Notation

<https://leetcode.com/problems/evaluate-reverse-polish-notation/>

def evalRPN(tokens):

    stack = []

    for token in tokens:

        if token == "+":

            stack.append(int(stack.pop()) + int(stack.pop()))

        elif token == "-":

            stack.append(-int(stack.pop()) + int(stack.pop()))

        elif token == "\*":

            stack.append(int(stack.pop()) \* int(stack.pop()))

        elif token == "/":

            a = int(stack.pop())

            b = int(stack.pop())

            stack.append(int(b / a))

        else:

            stack.append(token)

    return stack.pop()

tokens = ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

res = evalRPN(tokens)

print(res)

o/p

22

1. Generate Parenthesis

<https://leetcode.com/problems/generate-parentheses/>

def generateParanthesis(n):

    result = []

    res = ''

    def func(openP = 0, closeP = 0):

        nonlocal res

        if openP == n and closeP == n:

            result.append(res)

            return

        if openP < n:

            res = res + "("

            func(openP + 1, closeP)

            res = res[:-1]

        if closeP < openP:

            res = res + ")"

            func(openP, closeP + 1)

            res = res[:-1]

    func()

    return result

res = generateParanthesis(3)

print(res)

o/p

['((()))', '(()())', '(())()', '()(())', '()()()']

1. Daily Temperatures

<https://leetcode.com/problems/daily-temperatures/>

def dailyTemperatures(temps):

    stack = [[temps[0], 0]]

    res = [0] \* len(temps)

    for i,t in enumerate(temps):

        while stack and t > stack[-1][0]:

            res[stack[-1][1]] = i - stack[-1][1]

            stack.pop()

        stack.append([t, i])

    return res

temps = [73,74,75,71,69,72,76,73]

res = dailyTemperatures(temps)

print(res)

o/p

[1, 1, 4, 2, 1, 1, 0, 0]

1. Car Fleet

<https://leetcode.com/problems/car-fleet/>

def carFleet(target: int, position: list[int], speed: list[int]) -> int:

    posAndSpeed = [[p,s] for p,s in zip(position, speed)]

    posAndSpeed.sort(key=lambda x:x[0], reverse = True)

    stack = []

    for p,s in posAndSpeed:

        stack.append((target - p)/s)

        if len(stack) >= 2 and stack[-2] >= stack[-1]:

            stack.pop()

    return len(stack)

positions = [10,8,0,5,3]

speeds = [2,4,1,1,3]

target = 12

res = carFleet(target,positions, speeds)

print(res)

o/p

3

1. Largest Rectangle in histogram

<https://leetcode.com/problems/largest-rectangle-in-histogram/>

def largestRectangle(heights):

    maxArea = 0

    stack = []

    for i,h in enumerate(heights):

        start = i

        while stack and stack[-1][1] > h:

            index, height = stack.pop()

            maxArea = max(maxArea, height \* (i - index))

            start = index

        stack.append((start, h))

    for i, h in stack:

        maxArea = max(maxArea, h \* (len(heights) - i))

    return maxArea

heights = [2,1,5,6,2,3]

res = largestRectangle(heights)

print(res)

o/p

10

1. Next Greater Element I

<https://leetcode.com/problems/next-greater-element-i/description/>

def nextGreaterElement(nums1, nums2):

    hashMap = {n:i for i,n in enumerate(nums1)}

    res = [-1] \* len(nums1)

    stack = []

    for num in nums2:

        while stack and stack[-1] < num:

            smaller\_num = stack.pop()

            if smaller\_num in  hashMap:

                res[hashMap[smaller\_num]] = num

        stack.append(num)

    return res

o/p

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

1. Next Greater Element II  
   <https://leetcode.com/problems/next-greater-element-ii/>

def nextGreaterElement2(nums):

    N = len(nums)

    stack = []

    res = [-1] \* N \* 2

    for i in range(N \* 2):

        n =  nums[i % N]

        while stack and stack[-1][1] < n:

            ind, num = stack.pop()

            res[ind] = n

        stack.append([i,n])

    return res[:N]

n = [4,5,1,2,3,6,4,1]

res = nextGreaterElement2(n)

print(res)

o/p

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

Output: [5, 6, 2, 3, 6, -1, 5, 4]

1. Binary Search
2. Binary Search

<https://leetcode.com/problems/binary-search/>

def binSearch(arr, target):

    l,r = 0, len(arr) - 1

    while l <= r:

        mid = l + (r - l)//2

        if arr[mid] == target:

            return mid

        elif arr[mid] < target:

            l = mid + 1

        else:

            r = mid - 1

    return -1

arr = [1,2,6,4,7,9,11,15,18,21,25]

target = 18

res = binSearch(arr, target)

print(res)

o/p

8

1. Search in a 2D Matrix

<https://leetcode.com/problems/search-a-2d-matrix/description/>

def searchMatrix(matrix: list[list[int]], target: int) -> bool:

    def binSearch(arr):

        l,r = 0, len(arr) - 1

        while l <= r:

            mid = l + (r - l)//2

            if arr[mid] == target:

                return True

            elif arr[mid] < target:

                l = mid + 1

            else:

                r = mid - 1

        return False

    for arr in matrix:

        if binSearch(arr):

            return True

    return False

matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]]

target = 3

res = searchMatrix(matrix, target)

print(res)

o/p

True

1. Koko Eating Bananas

<https://leetcode.com/problems/koko-eating-bananas/description/>

from math import ceil

def minEatingSpeed(piles, h):

    maxPiles = max(piles)

    def checkHours(n):

        hours = 0

        for pile in piles:

            hours += ceil(pile/n)

        return hours

    l,r = 1, maxPiles

    speed = maxPiles

    while l <= r:

        mid = l + (r - l)//2

        s = checkHours(mid)

        if s > h:

            l = mid + 1

        elif s <= h:

            r = mid - 1

            speed = min(speed, mid)

    return speed

piles = [3,6,7,11]

h = 8

res =  minEatingSpeed(piles, h)

print(res)

o/p

4

1. Find Minimum in a Rotated Sorted Array

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

def findMinimum(nums):

    l,r = 0, len(nums) - 1

    minnum = nums[0]

    while l <= r:

        if nums[l] <= nums[r]:

            return min(minnum, nums[l])

        mid = l + (r - l) // 2

        minnum = min(minnum, nums[mid])

        if nums[mid] >= nums[l]:

            l = mid + 1

        else:

            r = mid - 1

arr = [4,5,6,8,10,11,-13,-12,-11,-10,-7,1,2,3]

res = search(arr)

print(res)

o/p

-13

1. Search in a Rotated Sorted Array

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

def func(nums, target):

    l,r = 0, len(nums) - 1

    while l <= r:

        mid = l + (r - l)//2

        if nums[mid] == target:

            return mid

        if nums[mid] > nums[l]:

            if nums[mid] > target >= nums[l]:

                r = mid - 1

            else:

                l = mid + 1

        else:

            if nums[mid] < target <= nums[r]:

                l = mid + 1

            else:

                r = mid - 1

    return -1

arr = [5,0,1,2,3,4]

res = func(arr, 0)

print(res)

o/p

-1

1. Search in a Rotated sorted Array II

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

def search(nums: list[int], target: int) -> bool:

    l,r = 0, len(nums) - 1

    while l <= r:

        mid = l + (r - l)//2

        if nums[mid] == target:

            return True

        if nums[mid] > nums[l]:

            if nums[mid] > target >= nums[l]:

                r = mid - 1

            else:

                l = mid + 1

        elif nums[mid] < nums[l]:

            if nums[mid] < target <= nums[r]:

                l = mid + 1

            else:

                r = mid - 1

        else:

            l += 1

    return False

arr = [2,5,6,0,0,1,2]

res = search(arr, 0)

print(res)

o/p

True

1. Time Based Key Value Storage

<https://neetcode.io/problems/time-based-key-value-store>

from collections import defaultdict

class TimeMap:

    def \_\_init\_\_(self):

*self*.hash = {}

    def get(self, key, timestamp):

        return *self*.getnextLeastValue(key, timestamp)

    def set(self, key, value, timestamp):

        if key not in *self*.hash:

*self*.hash[key] = []

*self*.hash[key].append([value, timestamp])

    def getnextLeastValue(self, key, timeStamp):

        values = *self*.hash.get(key, [])

        N = len(values)

        l,r = 0, N - 1

        res = ""

        while l<=r:

            mid = l + (r - l)//2

            if values[mid][1] == timeStamp:

                return values[mid][0]

            elif values[mid][1] < timeStamp:

                res = values[mid][0]

                l = mid + 1

            else:

                r = mid - 1

        return res

timeDict = TimeMap()

print(timeDict.set("foo", "bar", 1))

print(timeDict.get("foo", 1))

print(timeDict.get("foo", 3))

print(timeDict.set("foo", "bar2", 4))

print(timeDict.get("foo", 4))

print(timeDict.get("foo", 5))

o/p

None

bar

bar

None

bar2

bar2

1. Median of @ sorted Array

<https://leetcode.com/problems/median-of-two-sorted-arrays/>

def func(nums1, nums2):

    N1 = len(nums1)

    N2 = len(nums2)

    totalCount = N1 + N2

    half = totalCount//2

    A,B = nums1, nums2

    if N2 < N1:

        A,B = B,A

    l,r = 0, len(A) - 1

    while True:

        mid = l + (r - l)//2

# if mid == 4, that means, five elements, that is 0,1,2,3,4, so 5 elements must be subtracted from the half value, say if the half value is 10, 10 - (4 + 1) = 5, to get five elements in an array, we will consider till index 4, since that will give us 5 elements, 0,1,2,3,4. To get the index of the array 2 that will give us five elements is 10 - (4 + 1) - 1, i.e., half - mid - 2.

        mid2 = half - mid - 2

        ALeft = A[mid] if mid >= 0 else float("-infinity")

        ARight = A[mid + 1] if (mid + 1) < len(A) else float("infinity")

        BLeft = B[mid2] if mid2 >= 0 else float("-infinity")

        BRight = B[mid2 + 1] if (mid2 + 1) < len(B) else float("infinity")

        if ALeft <= BRight and BLeft <= ARight:

            if totalCount % 2:

                return min(ARight, BRight)

            return (max(ALeft, BLeft) + min(ARight, BRight)) / 2

        elif ALeft > BRight:

            r = mid - 1

        else:

            l = mid + 1

a1 = [1,2,3,4,5,6,7,8,9,9]

a2 = [3,4,5,6,7,8,9,10,12,13]

res = func(a1, a2)

print(res)

o/p

6.5