0001 Two sum

class Solution:

    def twoSum(self, nums, target):

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

            for j in range(i + 1, len(nums)):

                tmp = nums[i] + nums[j]

                if tmp == target:

                    return [i, j]

**0009 Palindrome number**

class Solution:

    def isPalindrome(self, x: int) -> bool:

        x = list(str(x))

        return x == x[::-1]

**0013 Roman to integer**

class Solution(object):

    def romanToInt(self, s):

        """

        :type s: str

        :rtype: int

        """

        grp = {'I' : 1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}

        sum = 0

        for i in range(0,len(s)-1):

            if grp[s[i]]>=grp[s[i+1]]:

                sum = sum+grp[s[i]]

            else :

                sum = sum-grp[s[i]]

        return sum+grp[s[-1]]

**0014 Longest common prefix**

class Solution:

    def longestCommonPrefix(self, strs: List[str]) -> str:

        strs, temp = sorted(strs),""

        for i in range(len(strs[0])):

            if strs[0][i] == strs[-1][i]: temp += strs[0][i]

            else: break

        return temp

**0020 Valid parentheses**

class Solution:

    def isValid(self, s: str) -> bool:

        stack=[]

        brckts={'}':'{',')':'(',']':'['}

        for brckt in s:

            if brckt in brckts.values():

                stack.append(brckt)

            else:

                if stack and brckts[brckt]==stack[-1] :

                    stack.pop()

                else:

                    return False

        if stack:

            return False

        return True

**0021 Merge Two Sorted Lists**

class Solution:

    def mergeTwoLists(self, lst1: Optional[ListNode], lst2: Optional[ListNode]) -> Optional[ListNode]:

        if lst1 is None or lst2 is None:

            return lst1 or lst2

        if lst1.val < lst2.val :

            lst1.next = self.mergeTwoLists(lst1.next, lst2)

            return lst1

        else:

            lst2.next = self.mergeTwoLists(lst1, lst2.next)

            return lst2

**0026 Remove Duplicates from Sorted Array**

class Solution:

    def removeDuplicates(self, num: List[int]) -> int:

        if len(num)==0: return 0

        i=0

        for j in range(1,len(num)):

            if num[i]!=num[j]:

                i+=1

                num[i]=num[j]

        return i+1

**0027 Remove Element**

class Solution:

    def removeElement(self, nums, val):

        for i in range(len(nums)):

            if nums[0] != val:

                nums.append(nums[0])

                del nums[0]

            else:

                del nums[0]

        return len(nums)

**0035 Search Insert Position**

class Solution:

    def searchInsert(self, nums: List[int], target: int) -> int:

        lft=0

        rgt=len(nums)-1

        while lft<=rgt:

            mid=(lft+rgt)//2

            if nums[mid]>target:

                rgt=mid-1

            elif nums[mid]<target:

                lft=mid+1

            else:

                return mid

        return lft

**0058 Length of Last Word**

class Solution:

    def lengthOfLastWord(self, s: str) -> int:

        lst = s.split(' ')

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

            if len(lst[i]) != 0:

                return (len(lst[i]))

        return 0

**0066 Plus One**

class Solution:

    def plusOne(self, digits: List[int]) -> List[int]:

        strint=str(int(''.join(map(str,digits)))+1)

        digits=list(map(int,list(strint)))

        return digits

**0067 Add Binary**

class Solution:

    def addBinary(self, a: str, b: str) -> str:

        a = int(a, 2)

        b = int(b, 2)

        return bin(a + b)[2:]

**0070 Climbing Stairs**

class Solution:

    def climbStairs(self, n: int) -> int:

        grp = [1, 1]

        for step in range(2, n+1):

            grp.append(grp[step-2] + grp[step-1])

        return grp[n]

**0083 Remove Duplicates from Sorted List**

class Solution:

    def deleteDuplicates(self, head: ListNode) -> ListNode:

        res = head

        def solve(rt):

            if rt :

                if rt.next != None and rt.val == rt.next.val :

                    rt.next = rt.next.next

                    solve(rt)

                else:

                    solve(rt.next)

        solve(head)

        return res

**0088 Merge Sorted Array**

class Solution:

    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:

         del nums1[m:]

         del nums2[n:]

         nums1 += nums2

         nums1.sort()

**0094 Binary Tree Inorder Traversal**

class Solution:

    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:

        output = []

        def dfs(node):

            if node:

                dfs(node.left)

                output.append(node.val)

                dfs(node.right)

        dfs(root)

        return output

**0101 Symmetric Tree**

class Solution:

    def isSymmetric(self, root: TreeNode) -> bool:

        if not root:

            return True

        return self.check(root.left, root.right)

    def check(self, left, right):

        if left is None and right is None:

            return True

        if left is None or right is None:

            return False

        if left.val != right.val:

            return False

        a = self.check(left.left, right.right)

        b = self.check(left.right, right.left)

        return a and b

**0104 Maximum Depth of Binary Tree**

class Solution:

    def maxDepth(self, root: Optional[TreeNode]) -> int:

        if root == None: return 0

        if root.left == None and root.right == None: return 1

        elif root.left == None: return self.maxDepth(root.right) + 1

        elif root.right == None: return self.maxDepth(root.left) + 1

        else: return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1

**0108 Convert Sorted Array to Binary Search Tree**

class Solution:

    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:

        def sortToBST(nums):

            if len(nums) == 0:

                return None

            mid = nums[len(nums) // 2]

            root = TreeNode(mid)

            root.left = sortToBST(nums[:len(nums) // 2])

            root.right = sortToBST(nums[len(nums) // 2 + 1:])

            return root

        return sortToBST(nums)

**0110 Balanced Binary Tree**

class Solution:

    ans = True

    def isBalanced(self, root: Optional[TreeNode]) -> bool:

        if not root:

            return True

        def dfs(node)->int:

            if not node:

                return -1

            l = dfs(node.left)

            r = dfs(node.right)

            if abs(l -r) > 1:

                self.ans = False

            return max(l, r) + 1

        dfs(root)

        return self.ans

**0111 Minimum Depth of Binary Tree**

class TreeNode:

    def \_\_init\_\_(self, x):

        self.val = x

        self.left = None

        self.right = None

class Solution:

    def minDepth(self, root):

        if root is None:

            return 0

        if root.left and root.right:

            return min(self.minDepth(root.left), self.minDepth(root.right)) + 1

        else:

            return max(self.minDepth(root.left), self.minDepth(root.right)) + 1

**0112 Path sum**

class Solution(object):

    def hasPathSum(self, root, targetSum):

        if not root:

            return False

        if not root.left and not root.right and root.val == targetSum:

            return True

        targetSum -= root.val

        return self.hasPathSum(root.left, targetSum) or self.hasPathSum(root.right, targetSum)

**0118 Pascal's triangle**

class Solution:

    def generate(self, numRows: int) -> List[List[int]]:

        ans = [[1]]

        for n in range(1, numRows):

            nextRow = [1]

            for m in range(n-1):

                nextRow.append(ans[n-1][m]+ans[n-1][m+1])

            nextRow.append(1)

            ans.append(nextRow)

        return ans

**0119 Pascal's Triangle II**

class Solution:

    def getRow(self, rowIndex: int) -> List[int]:

        DP =[0] \* (rowIndex+1)

        for i in range(len(DP)):

            DP[i] = [0] \* (i+1)

        DP[0][0] = 1

        for i in range(rowIndex+1) :

            for j in range(i+1) :

                if j ==0 or j == i :

                    DP[i][j] = 1

                else :

                    DP[i][j] = DP[i-1][j-1] + DP[i-1][j]

        return DP[rowIndex]

**0121 Best Time to Buy and Sell Stock**

class Solution:

    def maxProfit(self, prices: List[int]) -> int:

        profit = 0

        min\_price = sys.maxsize

        for price in prices:

            min\_price = min(min\_price, price)

            profit = max(profit, price - min\_price)

        return profit

**0136 Single Number**

class Solution:

    def singleNumber(self, nums: List[int]) -> int:

        s = set()

        for i in nums:

            if i not in s:

                s.add(i)

            else:

                s.remove(i)

        return list(s)[0]

**0141 Linked List Cycle**

class Solution:

    def hasCycle(self, head: Optional[ListNode]) -> bool:

        if not head or not head.next:   return False

        fast, slow = head, head

        while fast and fast.next:

            slow = slow.next

            fast = fast.next.next

            if fast == slow:

                return True

        return False

**0144 Binary Tree Preorder Traversal**

class Solution:

    def preorderTraversal(self, root: Optional[TreeNode]) -> List[int]:

        visit = []

        if root is not None:

            visit.append(root.val)

            visit.extend(self.preorderTraversal(root.left))

            visit.extend(self.preorderTraversal(root.right))

        return visit

**0145 Binary Tree Postorder Traversal**

class Solution:

    def dfs(self,root,res) :

        if root :

            self.dfs(root.left,res)

            self.dfs(root.right,res)

            res.append(root.val)

    def postorderTraversal(self, root: TreeNode) -> List[int]:

        res= []

        self.dfs(root,res)

        return res

**0160 Intersection of Two Linked Lists**

class Solution:

    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> Optional[ListNode]:

        a = headA

        b = headB

        while (a and b):

            if a == b: return a

            a = a.next

            b = b.next

            if not a and not b: return None

            if not a: a = headB

            if not b: b = headA

        return None

**0169 Majority Element**

class Solution(object):

    def majorityElement(self, nums):

        """

        :type nums: List[int]

        :rtype: int

        """

        count = collections.Counter(nums)

        return count.most\_common(1)[0][0]

**0171 Excel Sheet Column Number**

class Solution:

    def titleToNumber(self, columnTitle: str) -> int:

        ans = 0

        N = len(columnTitle)

        for n in range(N-1, -1, -1):

            ans += 26\*\*n \* (ord(columnTitle[N-n-1])-ord('A')+1)

        return ans

**0190 Reverse Bits**

class Solution:

    def reverseBits(self, n: int) -> int:

        binary\_str = (bin(n))[2:][::-1]

        ans\_str = binary\_str + "0"\*(32-len(binary\_str))

        ans = int(ans\_str, 2)

        return ans

**0191 Number of 1 Bits**

class Solution:

    def hammingWeight(self, n: int) -> int:

        return str(bin(n)).count('1')

**0202 Happy Number**

class Solution:

    def isHappy(self, n: int) -> bool:

        nSet = set()

        while n not in nSet:

            nSet.add(n)

            n = self.sumOfSquares(n)

            if n ==1 :

                return True

        return False

    def sumOfSquares(self, n:int) -> int:

        output = 0

        while n:

            digit = n%10

            digit = digit \*\*2

            output += digit

            n = n //10

        return output

**0203 Remove Linked List Elements**

class Solution:

    def removeElements(self, head: Optional[ListNode], val: int) -> Optional[ListNode]:

        node = ListNode(None)

        node.next = head

        head = node

        while head.next:

            if head.next.val==val:

                head.next=head.next.next

            else:

                head = head.next

        return node.next

**0206 Reverse Linked List**

class Solution:

    def reverseList(self, head: ListNode) -> ListNode:

        def reverse(node: ListNode, prev: ListNode = None ):

            if not node:

                return prev

            next, node.next = node.next, prev

            return reverse(next, node)

        return reverse(head)

**0217 Contains Duplicate**

class Solution:

    def containsDuplicate(self, nums: List[int]) -> bool:

        check\_table = {}

        for n in nums:

            if n in check\_table:

                return True

            else:

                check\_table[n] = 1

        return False

**0219 Contains Duplicate II**

class Solution:

    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:

        import collections

        nums\_map = collections.defaultdict(list)

        for i, num in enumerate(nums):

            nums\_map[num].append(i)

            if len(nums\_map[num]) >= 2:

                output = abs(nums\_map[num][-2] - i)

                if output <= k:

                    return True

        return False

**0222 Count Complete Tree Nodes**

def count(root: Optional[TreeNode]):

    if root == None: return 0;

    return count(root.left) + count(root.right) + 1

class Solution:

    def countNodes(self, root: Optional[TreeNode]) -> int:

        return count(root)

**0225 Implement Stack using Queues**

from collections import deque

class MyStack:

    def \_\_init\_\_(self):

        self.stack1 = deque()

        self.stack2 = deque()

    def push(self, x: int) -> None:

        self.stack1.append(x)

    def pop(self) -> int:

        while len(self.stack1) > 1:

            temp = self.stack1.popleft()

            self.stack2.append(temp)

        if len(self.stack1) == 1:

            answer = self.stack1.popleft()

            self.stack1, self.stack2 = self.stack2, self.stack1

        return answer

    def top(self) -> int:

        return self.stack1[-1]

    def empty(self) -> bool:

        return False if self.stack1 else True

**0226 Invert Binary Tree**

class Solution:

    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

        if root:

            root.left, root.right = self.invertTree(root.right), self.invertTree(root.left)

            return root

        return None

**0228 Summary Ranges**

class Solution:

    def summaryRanges(self, nums):

        if not nums:

            return []

        start = nums[0]

        end = nums[0]

        ans = []

        def beautify(a, b):

            if a == b:

                return str(a)

            else:

                return str(a) + "->" + str(b)

        for i in range(1,len(nums)):

            if (nums[i] - nums[i-1]) != 1:

                ans.append(beautify(start,nums[i-1]))

                start = nums[i]

        ans.append(beautify(start, nums[-1]))

        return ans

**0231 Power of Two**

class Solution:

    def isPowerOfTwo(self, n: int) -> bool:

        return n > 0 and n & (n-1) == 0

**0232 Implement Queue using Stacks**

class MyQueue:

    def \_\_init\_\_(self):

        self.stack\_in = list()

        self.stack\_out = list()

    def push(self, x: int) -> None:

        self.stack\_in.append(x)

    def pop(self) -> int:

        length = len(self.stack\_in) - 1

        for \_ in range(length):

            self.stack\_out.append(self.stack\_in.pop())

        front\_value = self.stack\_in.pop()

        for \_ in range(length):

            self.stack\_in.append(self.stack\_out.pop())

        return front\_value

    def peek(self) -> int:

        return self.stack\_in[0]

    def empty(self) -> bool:

        if self.stack\_in:

            return False

        return True

**0234 Palindrome Linked List**

class Solution:

    def isPalindrome(self, head: ListNode) -> bool:

        q: Deque = collections.deque()

        if not head:

            return True

        node = head

        while node is not None:

            q.append(node.val)

            node = node.next

        while len(q) > 1:

            if q.popleft() != q.pop():

                return False

        return True

**0242 Valid Anagram**

class Solution:

    def isAnagram(self, s: str, t: str) -> bool:

        return sorted(s) == sorted(t)

**0257 Binary Tree Paths**

class Solution:

    def binaryTreePaths(self, root: TreeNode) -> List[str]:

        def dfs(node, path):

            path += '->'

            path += str(node.val)

            if not node.left and not node.right:

                return ans.append(path[2:])

            if node.left:

                dfs(node.left, path)

            if node.right:

                dfs(node.right, path)

        ans = []

        dfs(root, "")

        return ans

**0258 Add Digits**

class Solution:

    def addDigits(self, num: int) -> int:

        while True:

            if num // 10 < 1:

                return num

            num\_list = list(map(int, list(str(num))))

            sum\_num = sum(num\_list)

            num = sum\_num

**0263 Ugly Number**

class Solution:

    def isUgly(self, num: int) -> bool:

        ans = False

        if num <= 0:

            return False

        nums = [2, 3, 5]

        for i in nums:

            while num % i == 0:

                num = num / i

        if num == 1:

            ans = True

        return ans

**0268 Missing Number**

class Solution:

    def missingNumber(self, nums: List[int]) -> int:

        n = len(nums)

        return int(((n \* (n+1)) / 2) - sum(nums))

**0283 Move Zeroes**

class Solution:

    def moveZeroes(self, nums):

        for i in range(len(nums))[::-1]:

            if nums[i] == 0:

                nums.pop(i)

                nums.append(0)

**0292 Nim Game**

class Solution:

  def canWinNim(self, n: int) -> bool:

    return n % 4 != 0

**0303 Range Sum Query – Immutable**

class NumArray:

  def \_\_init\_\_(self, nums: List[int]):

    self.prefix = [0] + list(itertools.accumulate(nums))

  def sumRange(self, left: int, right: int) -> int:

    return self.prefix[right + 1] - self.prefix[left]

**0338 Counting Bits**

class Solution:

    def countBits(self, n: int) -> List[int]:

        dp = [0]

        for i in range(1, n+1):

            dp.append(dp[i&(i-1)]+1)

        return dp

**0342 Power of Four**

class Solution:

  def isPowerOfFour(self, n: int) -> bool:

    return n > 0 and n.bit\_count() == 1 and (n - 1) % 3 == 0

**0344 Reverse String**

class Solution:

    def reverseString(self, s: List[str]) -> None:

        l = 0

        r = len(s) - 1

        while l < r:

            s[l], s[r] = s[r], s[l]

            l += 1

            r -= 1

**0345 Reverse Vowels of a String**

class Solution(object):

    def reverseVowels(self, s):

        """

        :type s: str

        :rtype: str

        """

        vowels = "aeiou"

        string = list(s)

        i, j = 0, len(s) - 1

        while i < j:

            if string[i].lower() not in vowels:

                i += 1

            elif string[j].lower() not in vowels:

                j -= 1

            else:

                string[i], string[j] = string[j], string[i]

                i += 1

                j -= 1

        return "".join(string)

**0349 Intersection of Two Arrays**

class Solution:

  def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:

    ans = []

    nums1 = set(nums1)

    for num in nums2:

      if num in nums1:

        ans.append(num)

        nums1.remove(num)

    return ans

**0350 Intersection of Two Arrays II**

class Solution:

    def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:

        a = set(nums1)

        b = set(nums2)

        s = a & b

        ans = []

        for num in s:

            count = min(nums1.count(num), nums2.count(num))

            ans.extend([num] \* count)

        return ans

**0374 Guess Number Higher or Lower**

class Solution:

    def guessNumber(self, n: int) -> int:

        left = 1

        right = n

        while left < right:

            mid = (left + right) // 2

            if guess(mid) == -1:

                right = mid - 1

            elif guess(mid) == 1:

                left = mid + 1

            else:

                return mid

        return left

**0383 Ransom Note**

class Solution:

    def canConstruct(self, ransomNote: str, magazine: str) -> bool:

        dict\_ran = collections.Counter(ransomNote)

        dict\_maz = collections.Counter(magazine)

        for key, value in dict\_ran.items():

            if key not in dict\_maz or value > dict\_maz[key]:

                return False

        return True

**0387 First Unique Character in a String**

class Solution:

    def firstUniqChar(self, s: str) -> int:

        d = {}

        for i in range(len(s)):

            if s[i] not in d: d[s[i]] = 0

            d[s[i]] += 1

        for i in range(len(s)):

            if d[s[i]] == 1:

                return i

        return -1

**0389 Find the Difference**

class Solution(object):

    def findTheDifference(self, s, t):

        """

        :type s: str

        :type t: str

        :rtype: str

        """

        x = 0

        for c in s+t:

            x ^= ord(c)

        return chr(x)

**0392 Is Subsequence**

class Solution:

    def isSubsequence(self, s: str, t: str) -> bool:

        len\_s = len(s)

        if len\_s == 0:

            return True

        cnt = 0

        for val in t:

            if val == s[cnt]:

                cnt = cnt + 1

                if cnt == len\_s:

                    return True

                    break

        return False

**0404 Sum of Left Leaves**

class Solution:

  def sumOfLeftLeaves(self, root: Optional[TreeNode]) -> int:

    if not root:

      return 0

    ans = 0

    if root.left:

      if not root.left.left and not root.left.right:

        ans += root.left.val

      else:

        ans += self.sumOfLeftLeaves(root.left)

    ans += self.sumOfLeftLeaves(root.right)

    return ans

**0409 Longest Palindrome**

class Solution:

    def longestPalindrome(self, s: str) -> int:

        ans = 0

        alpha = {}

        for c in s:

            if c not in alpha:

                alpha[c] = 1

            else:

                alpha[c] += 1

        odd\_exist = False

        max\_odd = 0

        for value in  alpha.values():

            if value % 2 == 0:

                ans += value

            else:

                max\_odd = max(max\_odd,value)

                odd\_exist = True

                ans += ( value -1 )

        if odd\_exist:

            ans += 1

        return ans

**0412 Fizz Buzz**

class Solution:

    def fizzBuzz(self, n: int) -> List[str]:

        ans = []

        for i in range(1,n+1):

            if i % 3 == 0 and i % 5 == 0:

                ans.append("FizzBuzz")

            elif i % 3 == 0:

                ans.append("Fizz")

            elif i % 5 == 0:

                ans.append("Buzz")

            else:

                ans.append(str(i))

        return ans

**0415 Add Strings**

class Solution:

  def addStrings(self, num1: str, num2: str) -> str:

    ans = []

    carry = 0

    i = len(num1) - 1

    j = len(num2) - 1

    while i >= 0 or j >= 0 or carry:

      if i >= 0:

        carry += int(num1[i])

      if j >= 0:

        carry += int(num2[j])

      ans.append(str(carry % 10))

      carry //= 10

      i -= 1

      j -= 1

    return ''.join(reversed(ans))

**0448 Find All Numbers Disappeared in an Array**

class Solution:

    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:

        set\_result = set(range(1,len(nums)+1))

        set\_nums = set(nums)

        return list(set\_result-set\_nums)

**0455 Assign Cookies**

class Solution:

    def findContentChildren(self, g: List[int], s: List[int]) -> int:

        g.sort()

        s.sort()

        i, j = 0,0

        ans = 0

        while i<len(g) and j<len(s):

            if g[i] <= s[j]:

                ans+=1

                i+=1

            j+=1

        return ans

**0461 Hamming Distance**

class Solution:

    def hammingDistance(self, x: int, y: int) -> int:

        return bin(x^y).count('1')

**0463 Island Perimeter**

class Solution:

    def islandPerimeter(self, grid: List[List[int]]) -> int:

        sqr = 0

        cl = 0

        for i in range(len(grid)):

            for j in range(len(grid[0])):

                if grid[i][j] == 1:

                    sqr += 1

                    if i == 0 and j > 0 and grid[0][j-1] == 1:

                        cl += 1

                    elif j == 0 and i > 0 and grid[i-1][0] == 1:

                        cl += 1

                    elif i != 0 and j != 0:

                        if grid[i-1][j] == 1: cl += 1

                        if grid[i][j-1] == 1: cl += 1

        return (sqr\*4) - (cl\*2)

**0476 Number Complement**

class Solution:

    def findComplement(self, num: int) -> int:

        ans = 0

        cur = 1

        while num > 1:

            a, b = num//2, num%2

            num = a

            if b == 0:

                ans += cur

            cur \*= 2

        return ans

**0485 Max Consecutive Ones**

class Solution:

    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:

        maximum = 0

        temp = 0

        for i in range(len(nums)):

            if nums[i] == 1:

                temp += 1

            else:

                maximum = max(maximum, temp)

                temp = 0

        return max(maximum, temp)

**0495 Teemo Attacking**

class Solution:

  def findPoisonedDuration(self, timeSeries: List[int], duration: int) -> int:

    if duration == 0:

      return 0

    ans = 0

    for i in range(0, len(timeSeries) - 1):

      ans += min(timeSeries[i + 1] - timeSeries[i], duration)

    return ans + duration

**0496 Next Greater Element I**

class Solution:

  def nextGreaterElement(self, nums1: List[int], nums2: List[int]) -> List[int]:

    numToNextGreater = {}

    stack = []

    for num in nums2:

      while stack and stack[-1] < num:

        numToNextGreater[stack.pop()] = num

      stack.append(num)

    return [numToNextGreater.get(num, -1) for num in nums1]

**0501 Find Mode in Binary Search Tree**

class Solution:

  def findMode(self, root: Optional[TreeNode]) -> List[int]:

    self.ans = []

    self.pred = None

    self.count = 0

    self.maxCount = 0

    def updateCount(root: Optional[TreeNode]) -> None:

      if self.pred and self.pred.val == root.val:

        self.count += 1

      else:

        self.count = 1

      if self.count > self.maxCount:

        self.maxCount = self.count

        self.ans = [root.val]

      elif self.count == self.maxCount:

        self.ans.append(root.val)

      self.pred = root

    def inorder(root: Optional[TreeNode]) -> None:

      if not root:

        return

      inorder(root.left)

      updateCount(root)

      inorder(root.right)

    inorder(root)

    return self.ans

**0506 Relative Ranks**

class Solution(object):

    def findRelativeRanks(self, nums):

        """

        :type nums: List[int]

        :rtype: List[str]

        """

        dic = {}

        size = len(nums)

        res = [None]\*size

        for i in range(size):

            dic[nums[i]] = i

        nums.sort()

        for i in range(size-1,-1,-1):

            if size-1-i == 0:

                res[dic[nums[i]]] = "Gold Medal"

            elif size-1-i == 1:

                res[dic[nums[i]]] = "Silver Medal"

            elif size-1-i == 2:

                res[dic[nums[i]]] = "Bronze Medal"

            else:

                res[dic[nums[i]]] = str(size-i)

        return res

**0520 Detect Capital**

class Solution:

    def detectCapitalUse(self, word: str) -> bool:

        return True if word.capitalize() == word or word.isupper() or word.islower() else False

**0521 Longest Uncommon Subsequence I**

class Solution:

  def findLUSlength(self, a: str, b: str) -> int:

    return -1 if a == b else max(len(a), len(b))

**0541 Reverse String II**

class Solution:

    def reverseStr(self, s, k):

        list\_s = list(s)

        for s in range(0, len(s), 2\*k):

            list\_s[s:s+k] = list\_s[s:s+k][::-1]

        return "".join(list\_s)

**0543 Diameter of Binary Tree**

class Solution(object):

    def diameterOfBinaryTree(self, root):

        """

        :type root: TreeNode

        :rtype: int

        """

        self.ans = 0

        def depth(p):

            if not p: return 0

            left, right = depth(p.left), depth(p.right)

            self.ans = max(self.ans, left+right)

            return 1 + max(left, right)

        depth(root)

        return self.ans

**0557 Reverse Words in a String III**

class Solution:

    def reverseWords(self, s: str) -> str:

        ans = ""

        for word in s.split(" "):

            ans += word[::-1] + ' '

        return ans[:-1]

**0561 Array Partition**

class Solution:

    def arrayPairSum(self, nums: List[int]) -> int:

        ans = 0

        nums.sort()

        for i in nums[0::2]:

            ans += i

        return ans

**0563 Binary Tree Tilt**

class Solution:

    def findTilt(self, root: TreeNode) -> int:

        total\_tilt = 0

        def dfs(node):

            nonlocal total\_tilt

            if not node:

                return 0

            left\_sum = dfs(node.left)

            right\_sum = dfs(node.right)

            tilt = abs(left\_sum - right\_sum)

            total\_tilt += tilt

            return left\_sum + right\_sum + node.val

        dfs(root)

        return total\_tilt

**0566 Reshape the Matrix**

class Solution:

  def matrixReshape(self, nums: List[List[int]], r: int, c: int) -> List[List[int]]:

    if nums == [] or r \* c != len(nums) \* len(nums[0]):

      return nums

    ans = [[0 for j in range(c)] for i in range(r)]

    k = 0

    for row in nums:

      for num in row:

        ans[k // c][k % c] = num

        k += 1

    return ans

**0575 Distribute Candies**

class Solution:

    def distributeCandies(self, candyType: List[int]) -> int:

            candyType.sort()

            num = int(len(candyType)/2)

            count = 1

            for i in range(len(candyType)-1):

                if candyType[i] != candyType[i+1]:

                    count+=1

            return num if count> num  else count

**0594 Longest Harmonious Subsequence**

class Solution:

  def findLHS(self, nums: List[int]) -> int:

    ans = 0

    count = collections.Counter(nums)

    for num, freq in count.items():

      if num + 1 in count:

        ans = max(ans, freq + count[num + 1])

    return ans

**0598 Range Addition II**

class Solution:

    def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:

        if len(ops) == 0 :

            return m \* n

        minrow = 10\*\*5

        mincol = 10\*\*5

        for i in range(len(ops)):

            minrow = min(ops[i][0],minrow)

            mincol = min(ops[i][1],mincol)

        return minrow \* mincol

**0599 Minimum Index Sum of Two Lists**

class Solution:

  def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:

    ans = []

    restaurantToIndex = {restaurant: i for i,

                         restaurant in enumerate(list1)}

    minSum = math.inf

    for i, restaurant in enumerate(list2):

      if restaurant in restaurantToIndex:

        summ = restaurantToIndex[restaurant] + i

        if summ < minSum:

          ans.clear()

        if summ <= minSum:

          ans.append(restaurant)

          minSum = summ

    return ans

**0617 Merge Two Binary Trees**

class Solution:

    def mergeTrees(self, root1: Optional[TreeNode], root2: Optional[TreeNode]) -> Optional[TreeNode]:

        if root1 and root2:

            node = TreeNode(root1.val + root2.val)

            node.left = self.mergeTrees(root1.left, root2.left)

            node.right = self.mergeTrees(root1.right, root2.right)

            return node

        else:

            return root1 or root2

**0637 Average of Levels in Binary Tree**

class Solution:

    def averageOfLevels(self, root: Optional[TreeNode]) -> List[float]:

        val = []

        def search(root, depth):

            nonlocal val

            if not root:

                return

            if len(val) < depth+1:

                val.append([])

            val[depth].append(root.val)

            search(root.left, depth+1)

            search(root.right, depth+1)

        search(root, 0)

        ans = [sum(x)/len(x) for x in val]

        return ans

**0645 Set Mismatch**

class Solution:

  def findErrorNums(self, nums: List[int]) -> List[int]:

    for num in nums:

      if nums[abs(num) - 1] < 0:

        duplicate = abs(num)

      else:

        nums[abs(num) - 1] \*= -1

    for i, num in enumerate(nums):

      if num > 0:

        return [duplicate, i + 1]

**0657 Robot Return to Origin**

from collections import Counter

class Solution:

    def judgeCircle(self, moves: str) -> bool:

        cnt = dict(Counter(moves))

        cnt\_keys = list(cnt.keys())

        pairs = [ {"L", "R"}, {"U", "D"}, {"L", "R", "U", "D"} ]

        if set(cnt\_keys) not in pairs:

            return False

        else:

            if set(cnt\_keys) == pairs[0]:

                if cnt['L'] != cnt['R']:

                    return False

            elif set(cnt\_keys) == pairs[1]:

                if cnt['U'] != cnt['D']:

                    return False

            elif set(cnt\_keys) == pairs[2]:

                if (cnt['L'] != cnt['R']) or (cnt['U'] != cnt['D']):

                    return False

        return True

**0661 Image Smoother**

class Solution:

  def imageSmoother(self, M: List[List[int]]) -> List[List[int]]:

    m = len(M)

    n = len(M[0])

    ans = [[0 for j in range(n)] for i in range(m)]

    for i in range(m):

      for j in range(n):

        ones = 0

        count = 0

        for y in range(max(0, i - 1), min(m, i + 2)):

          for x in range(max(0, j - 1), min(n, j + 2)):

            ones += M[y][x]

            count += 1

        ans[i][j] = ones // count

    return ans

**0674 Longest Continuous Increasing Subsequence**

class Solution:

  def findLengthOfLCIS(self, nums: List[int]) -> int:

    ans = 0

    j = 0

    for i in range(len(nums)):

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

        j = i

      ans = max(ans, i - j + 1)

    return ans

**0682 Baseball Game**

class Solution:

  def calPoints(self, operations: List[str]) -> int:

    ans = []

    for operation in operations:

      match operation:

        case '+':

          ans.append(ans[-1] + ans[-2])

        case 'D':

          ans.append(ans[-1] \* 2)

        case 'C':

          ans.pop()

        case default:

          ans.append(int(operation))

    return sum(ans)

**0696 Count Binary Substrings**

class Solution:

  def countBinarySubstrings(self, s: str) -> int:

    ans = 0

    prevCount = 0

    equals = 1

    for i in range(len(s) - 1):

      if s[i] == s[i + 1]:

        equals += 1

      else:

        ans += min(prevCount, equals)

        prevCount = equals

        equals = 1

    return ans + min(prevCount, equals)

**0697 Degree of an Array**

class Solution:

  def findShortestSubArray(self, nums: List[int]) -> int:

    ans = 0

    deg = 0

    deb = {}

    count = collections.Counter()

    for i, num in enumerate(nums):

      deb.setdefault(num, i)

      count[num] += 1

      if count[num] > deg:

        deg = count[num]

        ans = i - deb[num] + 1

      elif count[num] == deg:

        ans = min(ans, i - deb[num] + 1)

    return ans

**0724 Find Pivot Index**

class Solution:

    def pivotIndex(self, nums: List[int]) -> int:

        left = 0

        total = sum(nums)

        for i, x in enumerate(nums):

            if left == total - left - x:

                return i

            left += x

        return -1

**0728 Self Dividing Numbers**

class Solution:

    def selfDividingNumbers(self, left: int, right: int) -> List[int]:

        ans = [num for num in range(left, right+1) if '0' not in str(num) and all([num % int(n) == 0 for n in str(num)])]

        return ans

**0733 Flood Fill**

class Solution:

    def check(self, r, c, rlimit, climit):

        return 0 <= r < rlimit and 0 <= c < climit

    def floodFill(self, image: List[List[int]], sr: int, sc: int, newColor: int) -> List[List[int]]:

        startnum = image[sr][sc]

        image[sr][sc] = newColor

        q = []

        visit = []

        q.append((sr, sc))

        visit.append((sr, sc))

        dr, dc = [-1, 1, 0, 0], [0, 0, -1, 1]

        while q:

            qr, qc = q.pop()

            for dir in range(4):

                nr = qr + dr[dir]

                nc = qc + dc[dir]

                if (nr, nc) not in visit and self.check(nr, nc, len(image), len(image[0])):

                    if image[nr][nc] == startnum:

                        visit.append((nr, nc))

                        q.append((nr, nc))

                        image[nr][nc] = newColor

        return (image)

**0709 To Lower Case**

class Solution:

    def toLowerCase(self, s: str) -> str:

        ans = ""

        for i in s:

            if i.isupper():

                ans += i.lower()

            else:

                ans += i

        return ans

**0744 Find Smallest Letter Greater Than Target**

class Solution:

  def nextGreatestLetter(self, letters: List[str], target: str) -> str:

    l = bisect.bisect\_right(range(len(letters)), target,

                            key=lambda m: letters[m])

    return letters[l % len(letters)]

**0746 Min Cost Climbing Stairs**

from typing import List

class Solution:

    def minCostClimbingStairs(self, cost: List[int]) -> int:

        total = len(cost)

        min\_cost = [0] \* (total+1)

        for i in range(2, total + 1):

            one\_step = min\_cost[i-1] + cost[i-1]

            two\_step = min\_cost[i-2] + cost[i-2]

            min\_cost[i] = min(one\_step, two\_step)

        return min\_cost[total]

**0747 Largest Number At Least Twice of Others**

class Solution:

    def dominantIndex(self, nums: List[int]) -> int:

        max\_num = max(nums)

        check\_num = max\_num // 2

        check = [num for num in nums if check\_num < num]

        if len(set(check)) > 1:

            return -1

        else:

            return nums.index(max\_num)

**0748 Shortest Completing Word**

class Solution:

  def shortestCompletingWord(self, licensePlate: str, words: List[str]) -> str:

    def isMatch(word: str) -> bool:

      wordCount = collections.Counter(word)

      return False if any(wordCount[i] < count[i] for i in string.ascii\_letters) else True

    ans = '\*' \* 16

    count = collections.defaultdict(int)

    for c in licensePlate:

      if c.isalpha():

        count[c.lower()] += 1

    for word in words:

      if len(word) < len(ans) and isMatch(word):

        ans = word

    return ans

**0559 Maximum Depth of N-ary Tree**

class Solution:

  def maxDepth(self, root: 'Node') -> int:

    if not root:

      return 0

    if not root.children:

      return 1

    return 1 + max(self.maxDepth(child) for child in root.children)

**0766 Toeplitz Matrix**

class Solution:

    def isToeplitzMatrix(self, matrix: List[List[int]]) -> bool:

        H, W = len(matrix), len(matrix[0])

        for w in range(W-1):

            for h in range(H-1):

                if matrix[h][w] != matrix[h+1][w+1]:

                    return False

        return True

**0771 Jewels and Stones**

class Solution(object):

    def numJewelsInStones(self, J, S):

        J\_count = 0

        for x in S:

            if x in J:

                J\_count += 1

        return J\_count

**0700 Search in a Binary Search Tree**

class Solution:

    def searchBST(self, root: Optional[TreeNode], val: int) -> Optional[TreeNode]:

        while root:

            if root.val < val:

                if root.right:

                    root = root.right

                else:

                    return

            elif root.val > val:

                if root.left:

                    root = root.left

                else:

                    return

            else:

                return root

**0703 Kth Largest Element in a Stream**

class KthLargest:

    def \_\_init\_\_(self, k: int, nums: List[int]):

        self.k = k

        self.kth = sorted(nums, reverse=True)[:k]

    def add(self, val: int) -> int:

        self.kth.append(val)

        self.kth = sorted(self.kth, reverse=True)[:self.k]

        return self.kth[-1]

**0783 Minimum Distance Between BST Nodes**

class Solution:

    ans = sys.maxsize

    prev = -sys.maxsize

    def minDiffInBST(self, root: Optional[TreeNode]) -> int:

        if root.left:

            self.minDiffInBST(root.left)

        self.ans = min(self.ans, root.val - self.prev)

        self.prev = root.val

        if root.right:

            self.minDiffInBST(root.right)

        return self.ans

**0796 Rotate String**

class Solution:

    def rotateString(self, s: str, goal: str) -> bool:

        if len(s) != len(goal):

            return False

        for i in range(len(s)):

            if s == goal:

                return True

            s = s[1:] + s[0]

        return False

**0705 Design HashSet**

class MyHashSet:

    def \_\_init\_\_(self):

        self.hash = [0 for \_ in range(1000001)]

    def add(self, key: int) -> None:

        self.hash[key] = 1

    def remove(self, key: int) -> None:

        self.hash[key] = 0

    def contains(self, key: int) -> bool:

        return True if self.hash[key] else False

**0706 Design HashMap**

class MyHashMap:

    def \_\_init\_\_(self):

        self.hash = [-1 for \_ in range(1000001)]

    def put(self, key: int, value: int) -> None:

        self.hash[key] = value

    def get(self, key: int) -> int:

        return self.hash[key]

    def remove(self, key: int) -> None:

        self.hash[key] = -1

**0804 Unique Morse Code Words**

class Solution:

    def transformation(text):

        morse\_dict = {'a': '.-', 'b': '-...', 'c': '-.-.', 'd': '-..', 'e': '.', 'f': '..-.', 'g': '--.', 'h': '....', 'i': '..', 'j': '.---', 'k': '-.-', 'l': '.-..', 'm': '--', 'n': '-.', 'o': '---', 'p': '.--.', 'q': '--.-', 'r': '.-.', 's': '...', 't': '-', 'u': '..-', 'v': '...-', 'w': '.--', 'x': '-..-', 'y': '-.--', 'z': '--..'}

        char\_list = list(text)

        temp = ""

        for char in char\_list:

            temp = temp + morse\_dict[char]

        return temp

    def uniqueMorseRepresentations(self, words: List[str]) -> int:

        morse\_list = []

        for word in words:

            morse = self.transformation(word)

            morse\_list.append(morse)

        set\_morse = set(morse\_list)

        return len(set\_morse)

**0806 Number of Lines To Write String**

class Solution:

  def numberOfLines(self, widths: List[int], s: str) -> List[int]:

    numLines = 1

    runWidth = 0

    for c in s:

      width = widths[ord(c) - ord('a')]

      if runWidth + width <= 100:

        runWidth += width

      else:

        numLines += 1

        runWidth = width

    return [numLines, runWidth]

**0812 Largest Triangle Area**

class Solution:

  def largestTriangleArea(self, points: List[List[int]]) -> float:

    ans = 0

    for Ax, Ay in points:

      for Bx, By in points:

        for Cx, Cy in points:

          ans = max(ans, 0.5 \* abs((Bx - Ax) \* (Cy - Ay) -

                                   (Cx - Ax) \* (By - Ay)))

    return ans

**0819 Most Common Word**

class Solution:

    def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:

        words = [word for word in re.sub('[^\w]', ' ', paragraph).lower().split()

                 if word not in banned]

        counter = collections.Counter(words)

        return counter.most\_common(1)[0][0]

**0821 Shortest Distance to a Character**

class Solution:

    def shortestToChar(self, S: str, C: str) -> List[int]:

        s\_list = list(S)

        idx\_list = []

        for i in range(len(s\_list)):

            if s\_list[i] == C:

                idx\_list.append(i)

        ans\_list = []

        for i in range(len(s\_list)):

            temp = min([ abs(i-idx) for idx in idx\_list ])

            ans\_list.append(temp)

        return ans\_list

**0830 Positions of Large Groups**

class Solution:

  def largeGroupPositions(self, s: str) -> List[List[int]]:

    n = len(s)

    ans = []

    i = 0

    j = 0

    while i < n:

      while j < n and s[j] == s[i]:

        j += 1

      if j - i >= 3:

        ans.append([i, j - 1])

      i = j

    return ans

**0844 Backspace String Compare**

class Solution:

    def backspaceCompare(self, s: str, t: str) -> bool:

        s\_que, t\_que = deque(s), deque(t)

        s\_stack, t\_stack = [], []

        while s\_que:

            s1 = s\_que.popleft()

            if s1!='#':

                s\_stack.append(s1)

            else:

                if len(s\_stack)!=0:

                    s\_stack.pop()

        while t\_que:

            t1 = t\_que.popleft()

            if t1!='#':

                t\_stack.append(t1)

            else:

                if len(t\_stack)!=0:

                    t\_stack.pop()

        return True if s\_stack==t\_stack else False

**0860 Lemonade Change**

class Solution(object):

    def lemonadeChange(self, bills):

        five, ten = 0, 0

        for dollar in bills:

            if dollar == 5:

                five += 1

            elif dollar == 10:

                if five == 0:

                    return False

                five -= 1

                ten += 1

            else:

                if five > 0 and ten > 0:

                    five -= 1

                    ten -= 1

                elif five >= 3:

                    five -= 3

                else:

                    return False

        return True

**0867 Transpose Matrix**

class Solution:

    def transpose(self, matrix: List[List[int]]) -> List[List[int]]:

        H, W = len(matrix), len(matrix[0])

        ans = [[0 for \_ in range(H)] for \_ in range(W)]

        for h in range(H):

            for w in range(W):

                ans[w][h] = matrix[h][w]

        return ans

**0868 Binary Gap**

class Solution:

  def binaryGap(self, n: int) -> int:

    ans = 0

    d = -32

    while n:

      if n & 1:

        ans = max(ans, d)

        d = 0

      n //= 2

      d += 1

    return ans

**0872 Leaf-Similar Trees**

class Solution:

    def leafSimilar(self, root1: Optional[TreeNode], root2: Optional[TreeNode]) -> bool:

        leaf1, leaf2 = [], []

        def get\_leaves(root, leaf):

            if not root:

                return

            if root.left:

                get\_leaves(root.left, leaf)

            if root.right:

                get\_leaves(root.right, leaf)

            if not root.left and not root.right:

                leaf.append(root.val)

        get\_leaves(root1, leaf1)

        get\_leaves(root2, leaf2)

        return True if leaf1 == leaf2 else False

**0876 Middle of the Linked List**

class Solution:

    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:

        N = 0

        nhead = head

        while nhead.next:

            nhead = nhead.next

            N += 1

        half = N//2 + N%2

        while half > 0:

            head = head.next

            half -= 1

        return head

**0883 Projection Area of 3D Shapes**

class Solution:

  def projectionArea(self, grid: List[List[int]]) -> int:

    return sum(a > 0 for row in grid for a in row) + sum(max(row) for row in grid) + sum(max(col) for col in zip(\*grid))

**0884 Uncommon Words from Two Sentences**

from collections import Counter

class Solution:

    def uncommonFromSentences(self, A: str, B: str) -> List[str]:

        array = A.split(" ") + B.split(" ")

        cnt = list(Counter(array).items())

        ans = [item[0] for item in cnt if item[1] == 1]

        return ans

**0892 Surface Area of 3D Shapes**

class Solution:

  def surfaceArea(self, grid: List[List[int]]) -> int:

    ans = 0

    for i in range(len(grid)):

      for j in range(len(grid)):

        if grid[i][j]:

          ans += grid[i][j] \* 4 + 2

        if i > 0:

          ans -= min(grid[i][j], grid[i - 1][j]) \* 2

        if j > 0:

          ans -= min(grid[i][j], grid[i][j - 1]) \* 2

    return ans

**0896 Monotonic Array**

class Solution:

    def isMonotonic(self, A: List[int]) -> bool:

        reverse\_flag = False

        if max(A) == A[0]:

            reverse\_flag = True

        sorted\_A = sorted(A, reverse=reverse\_flag)

        if sorted\_A == A:

            return True

        return False

**0897 Increasing Order Search Tree**

class Solution:

  def increasingBST(self, root: TreeNode, tail: TreeNode = None) -> TreeNode:

    if not root:

      return tail

    res = self.increasingBST(root.left, root)

    root.left = None

    root.right = self.increasingBST(root.right, tail)

    return res

**0905 Sort Array By Parity**

class Solution:

    def sortArrayByParity(self, nums: List[int]) -> List[int]:

        return sorted(nums, key=lambda x: x%2)

**0908 Smallest Range I**

class Solution:

    def smallestRangeI(self, A: List[int], K: int) -> int:

        return max(max(A) - min(A) - 2 \* K, 0)

**0917 Reverse Only Letters**

class Solution:

    def reverseOnlyLetters(self, S: str) -> str:

        temp = []

        indexs = {}

        ans = []

        cnt = 0

        s\_list = list(S)

        for i in range(len(s\_list)):

            if s\_list[i].isalpha() == True:

                temp.append(s\_list[i])

                cnt = cnt + 1

            if s\_list[i].isalpha() == False:

                indexs[i] = s\_list[i]

        if cnt == len(S):

            ans = S[::-1]

        else:

            keys = list(indexs.keys())

            temp = temp[::-1]

            idx = 0

            for i in range(len(S)):

                if i in keys:

                    ans.append(indexs[i])

                elif i not in keys:

                    ans.append(temp[idx])

                    idx = idx + 1

            ans = ''.join(ans)

        return ans

**0922 Sort Array By Parity II**

class Solution:

  def sortArrayByParityII(self, nums: List[int]) -> List[int]:

    n = len(nums)

    i = 0

    j = 1

    while i < n:

      while i < n and nums[i] % 2 == 0:

        i += 2

      while j < n and nums[j] % 2 == 1:

        j += 2

      if i < n:

        nums[i], nums[j] = nums[j], nums[i]

    return nums

**0929 Unique Email Addresses**

class Solution:

    def numUniqueEmails(self, emails: List[str]) -> int:

        res = set()

        for i in range(len(emails)):

            splitidx = emails[i].index('@')

            local = emails[i][0:splitidx]

            domain = emails[i][splitidx:]

            local = local.replace('.', '')

            if '+' in local:

                local = local[0: local.index('+')]

            address = local + domain

            res.add(address)

        return (len(res))

**0933 Number of Recent Calls**

class RecentCounter:

  def \_\_init\_\_(self):

    self.q = collections.deque()

  def ping(self, t: int) -> int:

    self.q.append(t)

    while self.q[0] < t - 3000:

      self.q.popleft()

    return len(self.q)

**0941 Valid Mountain Array**

class Solution:

    def validMountainArray(self, arr: List[int]) -> bool:

        ans = False

        valid = False

        n = len(arr)

        i = 0

        while i + 1 < n and arr[i] < arr[i+1]:

            i += 1

            valid = True

        while i + 1 < n and arr[i] > arr[i+1] and valid == True:

            i += 1

            ans = True

        if i+1 != n :

            ans = False

        return ans

**0942 DI String Match**

class Solution:

  def diStringMatch(self, s: str) -> List[int]:

    ans = []

    less = 0

    more = len(s)

    for c in s:

      if c == 'I':

        ans.append(less)

        less += 1

      else:

        ans.append(more)

        more -= 1

    return ans + [less]

**0944 Delete Columns to Make Sorted**

class Solution:

  def minDeletionSize(self, strs: List[str]) -> int:

    ans = 0

    for j in range(len(strs[0])):

      for i in range(len(strs) - 1):

        if strs[i][j] > strs[i + 1][j]:

          ans += 1

          break

    return ans

**0961 N-Repeated Element in Size 2N Array**

class Solution:

  def repeatedNTimes(self, nums: List[int]) -> int:

    for i in range(len(nums) - 2):

      if nums[i] == nums[i + 1] or nums[i] == nums[i + 2]:

        return nums[i]

    return nums[-1]

**0965 Univalued Binary Tree**

class Solution:

  def isUnivalTree(self, root: Optional[TreeNode]) -> bool:

    if not root:

      return True

    if root.left and root.left.val != root.val:

      return False

    if root.right and root.right.val != root.val:

      return False

    return self.isUnivalTree(root.left) and self.isUnivalTree(root.right)

**0509 Fibonacci Number**

class Solution:

    def fib(self, n: int) -> int:

        fi = [0]\*31

        fi[1] = 1

        for idx in range(2, n+1):

            fi[idx] = fi[idx-1] + fi[idx-2]

        return fi[n]

**0976 Largest Perimeter Triangle**

class Solution:

  def largestPerimeter(self, nums: List[int]) -> int:

    nums = sorted(nums)

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

      if nums[i - 2] + nums[i - 1] > nums[i]:

        return nums[i - 2] + nums[i - 1] + nums[i]

    return 0

**0989 Add to Array-Form of Integer**

class Solution:

  def addToArrayForm(self, num: List[int], k: int) -> List[int]:

    for i in reversed(range(len(num))):

      k, num[i] = divmod(num[i] + k, 10)

    while k > 0:

      num = [k % 10] + num

      k //= 10

    return num

**0997 Find the Town Judge**

class Solution:

    def findJudge(self, n: int, trust: List[List[int]]) -> int:

        if not trust:

            if n == 1:

                return 1

            else:

                return -1

        rT = [[] for \_ in range(n+1)]

        for a, b in trust:

            rT[b].append(a)

        for i in range(1, n+1):

            if len(rT[i]) == n-1:

                for t in range(1, n+1):

                    if i in rT[t]:

                        return -1

                return i

        return -1

**1002 Find Common Characters**

class Solution:

  def commonChars(self, words: List[str]) -> List[str]:

    return functools.reduce(lambda a, b: a & b,

                            map(collections.Counter, words)).elements()

**1005 Maximize Sum Of Array After K Negations**

class Solution:

  def largestSumAfterKNegations(self, nums: List[int], k: int) -> int:

    nums.sort()

    for i, num in enumerate(nums):

      if num > 0 or k == 0:

        break

      nums[i] = -num

      k -= 1

    return sum(nums) - (k % 2) \* min(nums) \* 2

**1009 Complement of Base 10 Integer**

class Solution:

    def bitwiseComplement(self, N: int) -> int:

        return int("".join(list(map(str, [1-num for num in list(map(int, list(str(bin(N))[2:])))]))), 2)

**1013 Partition Array Into Three Parts With Equal Sum**

class Solution:

  def canThreePartsEqualSum(self, arr: List[int]) -> bool:

    summ = sum(arr)

    if summ % 3 != 0:

      return False

    average = summ // 3

    ptCount = 0

    ptSum = 0

    for a in arr:

      ptSum += a

      if ptSum == average:

        ptCount += 1

        ptSum = 0

    return ptCount >= 3

**1018 Binary Prefix Divisible By 5**

class Solution:

  def prefixesDivBy5(self, nums: List[int]) -> List[bool]:

    ans = []

    cur = 0

    for num in nums:

      cur = (cur \* 2 + num) % 5

      ans.append(cur % 5 == 0)

    return ans

**1021 Remove Outermost Parentheses**

class Solution:

  def removeOuterParentheses(self, s: str) -> str:

    ans = []

    opened = 0

    for c in s:

      if c == '(':

        opened += 1

        if opened > 1:

          ans.append(c)

      else:  # c == ')'

        opened -= 1

        if opened > 0:

          ans.append(c)

    return ''.join(ans)

**1022 Sum of Root To Leaf Binary Numbers**

class Solution:

  def sumRootToLeaf(self, root: Optional[TreeNode]) -> int:

    ans = 0

    def dfs(root: Optional[TreeNode], val: int) -> None:

      nonlocal ans

      if not root:

        return

      val = val \* 2 + root.val

      if not root.left and not root.right:

        ans += val

      dfs(root.left, val)

      dfs(root.right, val)

    dfs(root, 0)

    return ans

**1025 Divisor Game**

class Solution:

    def divisorGame(self, n: int) -> bool:

        ans = [False]

        ans.append(False)

        for i in range(2, n+1):

            ans.append(False)

            for j in range(i-1, 0, -1):

                if i % j == 0:

                    if ans[i-j] == False:

                        ans[i] = True

                        break

        return ans[n]

**1030 Matrix Cells in Distance Order**

class Solution:

  def allCellsDistOrder(self, rows: int, cols: int, rCenter: int, cCenter: int) -> List[List[int]]:

    dirs = ((0, 1), (1, 0), (0, -1), (-1, 0))

    ans = []

    q = collections.deque([(rCenter, cCenter)])

    seen = {(rCenter, cCenter)}

    while q:

      i, j = q.popleft()

      ans.append([i, j])

      for dx, dy in dirs:

        x = i + dx

        y = j + dy

        if x < 0 or x == rows or y < 0 or y == cols:

          continue

        if (x, y) in seen:

          continue

        seen.add((x, y))

        q.append((x, y))

    return ans

**1160 Find Words That Can Be Formed by Characters**

class Solution:

  def countCharacters(self, words: List[str], chars: str) -> int:

    ans = 0

    count = collections.Counter(chars)

    for word in words:

      tempCount = count.copy()

      for c in word:

        tempCount[c] -= 1

        if tempCount[c] < 0:

          ans -= len(word)

          break

      ans += len(word)

    return ans

**1037 Valid Boomerang**

class Solution:

  def isBoomerang(self, points: List[List[int]]) -> bool:

    return (points[1][0] - points[0][0]) \* (points[2][1] - points[1][1]) != \

        (points[1][1] - points[0][1]) \* (points[2][0] - points[1][0])

**1046 Last Stone Weight**

class Solution:

    def lastStoneWeight(self, stones: List[int]) -> int:

        while len(stones) > 1:

            stones.sort(reverse=True)

            if stones[0] > stones[1]:

                stones.append(stones[0]-stones[1])

            del stones[1]

            del stones[0]

        if stones:

            return stones[0]

        else:

            return 0

**1051 Height Checker**

class Solution:

    def heightChecker(self, heights: List[int]) -> int:

        sorted\_height = sorted(heights)

        cnt = 0

        for i, height in enumerate(heights):

            if sorted\_height[i] != height:

                cnt = cnt + 1

        return cnt

**1071 Greatest Common Divisor of Strings**

class Solution:

  def gcdOfStrings(self, str1: str, str2: str) -> str:

    def mod(s1: str, s2: str) -> str:

      while s1.startswith(s2):

        s1 = s1[len(s2):]

      return s1

    if len(str1) < len(str2):

      return self.gcdOfStrings(str2, str1)

    if not str1.startswith(str2):

      return ''

    if not str2:

      return str1

    return self.gcdOfStrings(str2, mod(str1, str2))

**1089 Duplicate Zeros**

class Solution:

    def duplicateZeros(self, arr: List[int]) -> None:

        temp = []

        for i in range(len(arr)):

            if arr[i] == 0:

                temp.append(arr[i])

            temp.append(arr[i])

        ans = temp[:len(arr)]

        for i in range(len(arr)):

            arr[i] = ans[i]

**1103 Distribute Candies to People**

class Solution:

  def distributeCandies(self, candies: int, n: int) -> List[int]:

    ans = [0] \* n

    rows = int((-n + (n\*\*2 + 8 \* n\*\*2 \* candies)\*\*0.5) / (2 \* n\*\*2))

    accumN = rows \* (rows - 1) \* n // 2

    for i in range(n):

      ans[i] = accumN + rows \* (i + 1)

    givenCandies = (n\*\*2 \* rows\*\*2 + n \* rows) // 2

    candies -= givenCandies

    lastGiven = rows \* n

    i = 0

    while candies > 0:

      lastGiven += 1

      actualGiven = min(lastGiven, candies)

      candies -= actualGiven

      ans[i] += actualGiven

      i += 1

    return ans

**1108 Defanging an IP Address**

class Solution:

    def defangIPaddr(self, address: str) -> str:

        return address.replace('.', '[.]')

**1619 Mean of Array After Removing Some Elements**

class Solution:

    def trimMean(self, arr: List[int]) -> float:

        sorted\_arr = sorted(arr)

        five\_percent = int(len(arr)\*0.05)

        arr\_len = len(arr)

        ans\_arr = sorted\_arr[five\_percent:arr\_len-five\_percent]

        return sum(ans\_arr) / len(ans\_arr)

**1122 Relative Sort Array**

from collections import Counter

class Solution:

    def relativeSortArray(self, arr1, arr2):

        tmp1 = []

        tmp2 = []

        cnt = Counter(arr1)

        for num in arr2:

            if num in arr1:

                tmp1 = tmp1 + [num] \* cnt[num]

        tmp2 = sorted([num for num in arr1 if num not in arr2 ])

        return tmp1 + tmp2

**1287 Element Appearing More Than 25% In Sorted Array**

from collections import Counter

class Solution:

    def findSpecialInteger(self, arr: List[int]) -> int:

        ans = 0

        percent\_25\_num = int(len(arr) // 4)

        cnt = Counter(arr)

        keys = list(cnt.keys())

        for key in keys:

            if cnt[key] > percent\_25\_num:

                ans = key

                break

        return ans

**1128 Number of Equivalent Domino Pairs**

class Solution:

  def numEquivDominoPairs(self, dominoes: List[List[int]]) -> int:

    ans = 0

    count = collections.Counter()

    for domino in dominoes:

      key = min(domino[0], domino[1]) \* 10 + max(domino[0], domino[1])

      ans += count[key]

      count[key] += 1

    return ans

**1299 Replace Elements with Greatest Element on Right Side**

class Solution:

    def replaceElements(self, arr: List[int]) -> List[int]:

        ans = [0] \* len(arr)

        ans[-1] = -1

        for i in range(len(arr)-2, -1, -1):

            ans[i] = max([ans[i+1], arr[i+1]])

        return ans

**1137 N-th Tribonacci Number**

class Solution:

    def tribonacci(self, n: int) -> int:

        dp = [0] \* (38)

        dp[0] = 0

        dp[1] = 1

        dp[2] = 1

        for i in range(3,n+1):

            dp[i] = dp[i-1] + dp[i-2] + dp[i-3]

        return dp[n]

for i in range(0,26):

    print(Solution().tribonacci(i))

**1313 Decompress Run-Length Encoded List**

class Solution:

  def decompressRLElist(self, nums: List[int]) -> List[int]:

    ans = []

    for i in range(0, len(nums), 2):

      ans += [nums[i + 1]] \* nums[i]

    return ans

**1331 Rank Transform of an Array**

class Solution:

  def arrayRankTransform(self, arr: List[int]) -> List[int]:

    rank = {}

    for a in sorted(arr):

      if a not in rank:

        rank[a] = len(rank) + 1

    return map(rank.get, arr)

**1154 Day of the Year**

class Solution:

    def dayOfYear(self, date: str) -> int:

        year, month, day = int(date.split('-')[0]), int(date.split('-')[1]), int(date.split('-')[2])

        days = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

        ans = sum(days[:month-1]) + day

        if ((year % 4 == 0 and year % 100 != 0) or year % 400 == 0) and month > 2:

            ans = ans + 1

        return ans

**1935 Maximum Number of Words You Can Type**

class Solution:

  def canBeTypedWords(self, text: str, brokenLetters: str) -> int:

    ans = 0

    broken = set(brokenLetters)

    for word in text.split():

      ans += all(c not in broken for c in word)

    return ans

**1360 Number of Days Between Two Dates**

import datetime

class Solution:

    def daysBetweenDates(self, date1: str, date2: str) -> int:

        date\_01 = datetime.datetime.strptime(date1, '%Y-%m-%d')

        date\_02 = datetime.datetime.strptime(date2, '%Y-%m-%d')

        ans = abs(date\_01- date\_02).days

        return ans

**1175 Prime Arrangements**

class Solution:

  def numPrimeArrangements(self, n: int) -> int:

    kMod = 1\_000\_000\_007

    def countPrimes(n: int) -> int:

      isPrime = [False] \* 2 + [True] \* (n - 1)

      for i in range(2, int(n\*\*0.5) + 1):

        if isPrime[i]:

          for j in range(i \* i, n + 1, i):

            isPrime[j] = False

      return sum(isPrime)

    def factorial(n: int) -> int:

      fact = 1

      for i in range(1, n + 1):

        fact = fact \* i % kMod

      return fact

    count = countPrimes(n)

    return factorial(count) \* factorial(n - count) % kMod

**1507 Reformat Date**

class Solution:

    def reformatDate(self, date: str) -> str:

        month\_dict = {"Jan":"01", "Feb":"02", "Mar":"03", "Apr":"04", "May":"05", "Jun":"06", "Jul":"07", "Aug":"08", "Sep":"09", "Oct":"10", "Nov":"11", "Dec":"12"}

        split\_date = date.split(" ")

        day = split\_date[0][:-2]

        if len(day) == 1:

            day = "0" + day

        ans = "{}-{}-{}".format(split\_date[2], month\_dict[split\_date[1]], day)

        return ans

**1184 Distance Between Bus Stops**

class Solution:

  def distanceBetweenBusStops(self, distance: List[int], start: int, destination: int) -> int:

    clock = 0

    counter = 0

    if start > destination:

      start, destination = destination, start

    for i, d in enumerate(distance):

      if i >= start and i < destination:

        clock += d

      else:

        counter += d

    return min(clock, counter)

**1185 Day of the Week**

import datetime

class Solution:

    def dayOfTheWeek(self, day: int, month: int, year: int) -> str:

        week\_dict = {0:"Monday", 1:"Tuesday", 2:"Wednesday", 3:"Thursday", 4:"Friday", 5:"Saturday", 6:"Sunday"}

        weekday = datetime.datetime(year, month, day).weekday()

        return week\_dict[weekday]

**1550 Three Consecutive Odds**

class Solution:

    def threeConsecutiveOdds(self, arr: List[int]) -> bool:

        cnt = 0

        for num in arr:

            if num % 2 == 1:

                cnt = cnt + 1

                if cnt == 3:

                    return True

            else:

                cnt = 0

        return False

**1189 Maximum Number of Balloons**

from collections import Counter

class Solution:

    def maxNumberOfBalloons(self, text: str) -> int:

        cnt = collections.Counter(text)

        ballon\_key = ["b", "a", "l", "o", "n"]

        for key in ballon\_key:

            if key not in list(cnt.keys()):

                return 0

        return min(cnt['b'],cnt['a'],cnt['l']//2,cnt['o']//2,cnt['n'])

**1957 Delete Characters to Make Fancy String**

class Solution:

  def makeFancyString(self, s: str) -> str:

    ans = []

    for c in s:

      if len(ans) < 2 or ans[-1] != c or ans[-2] != c:

        ans.append(c)

    return ''.join(ans)

**1200 Minimum Absolute Difference**

class Solution:

    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:

        arrSorted = sorted(arr)

        ans = []

        mini = arrSorted[-1] - arrSorted[0]

        for n in range(len(arr)-1):

            mini = min(mini, arrSorted[n+1] - arrSorted[n])

        for n in range(len(arr)-1):

            if arrSorted[n+1] - arrSorted[n] == mini:

                ans.append([arrSorted[n], arrSorted[n+1]])

        return ans

**1207 Unique Number of Occurrences**

class Solution:

    def uniqueOccurrences(self, arr: List[int]) -> bool:

        value = Counter(arr).values()

        return True if sorted(value) == sorted(set(value)) else False

**1217 Minimum Cost to Move Chips to The Same Position**

class Solution:

  def minCostToMoveChips(self, chips: List[int]) -> int:

    count = [0] \* 2

    for chip in chips:

      count[chip % 2] += 1

    return min(count[0], count[1])

**1221 Split a String in Balanced Strings**

class Solution:

    def balancedStringSplit(self, s: str) -> int:

        length = len(s)

        ans = 0

        p = 0

        for i in range(length) :

            if s[p:i+1].count('L') == s[p:i+1].count('R') :

                ans += 1

                p = i+1

        return ans

**1232 Check If It Is a Straight Line**

class Solution:

    def checkStraightLine(self, coordinates: List[List[int]]) -> bool:

        x\_points = [x[0] for x in coordinates]

        if len(set(x\_points)) == 1:

            return True

        else:

            point1\_x, point1\_y = coordinates[0]

            point2\_x, point2\_y = coordinates[1]

            dy = (point1\_y-point2\_y)

            dx = (point1\_x-point2\_x)

            if dx == 0:

                dx = 1

            bias = point1\_y - dy/dx \* point1\_x

            for i in range(2,len(coordinates)):

                point\_x, point\_y = coordinates[i]

                if point\_y != dy/dx\* point\_x + bias:

                    return False

        return True

**2273 Find Resultant Array After Removing Anagrams**

class Solution:

  def removeAnagrams(self, words: List[str]) -> List[str]:

    ans = []

    def isAnagram(a: str, b: str) -> bool:

      count = collections.Counter(a)

      count.subtract(collections.Counter(b))

      return all(value == 0 for value in count.values())

    i = 0

    while i < len(words):

      j = i + 1

      while j < len(words) and isAnagram(words[i], words[j]):

        j += 1

      ans.append(words[i])

      i = j

    return ans

**2309 Greatest English Letter in Upper and Lower Case**

class Solution:

  def greatestLetter(self, s: str) -> str:

    seen = set(s)

    for i in range(25, -1, -1):

      if chr(ord('a') + i) in seen and \

              chr(ord('A') + i) in seen:

        return chr(ord('A') + i)

    return ''

**1252 Cells with Odd Values in a Matrix**

class Solution:

  def oddCells(self, m: int, n: int, indices: List[List[int]]) -> int:

    rows = [False] \* m

    cols = [False] \* n

    for r, c in indices:

      rows[r] ^= True

      cols[c] ^= True

    return sum(rows[i] ^ cols[j]

               for i in range(m)

               for j in range(n))

**2303 Calculate Amount Paid in Taxes**

class Solution:

  def calculateTax(self, brackets: List[List[int]], income: int) -> float:

    ans = 0

    prev = 0

    for upper, percent in brackets:

      if income < upper:

        return ans + (income - prev) \* percent / 100.0

      ans += (upper - prev) \* percent / 100.0

      prev = upper

    return ans

**1260 Shift 2D Grid**

class Solution:

    def shiftGrid(self, grid: List[List[int]], k: int) -> List[List[int]]:

        H, W = len(grid), len(grid[0])

        ngrid = [[0 for \_ in range(W)] for \_ in range(H)]

        k %= (H\*W)

        for h in range(H):

            for w in range(W):

                temp = h\*W + w + k

                dh, dw = temp//W, temp%W

                ngrid[dh%H][dw%W] = grid[h][w]

        return ngrid

**2215 Find the Difference of Two Arrays**

class Solution:

  def findDifference(self, nums1: List[int], nums2: List[int]) -> List[List[int]]:

    set1 = set(nums1)

    set2 = set(nums2)

    return [set1 - set2, set2 - set1]

**1266 Minimum Time Visiting All Points**

class Solution:

  def minTimeToVisitAllPoints(self, points: List[List[int]]) -> int:

    ans = 0

    for i in range(1, len(points)):

      ans += max(abs(points[i][0] - points[i - 1][0]),

                 abs(points[i][1] - points[i - 1][1]))

    return ans

**1275 Find Winner on a Tic Tac Toe Game**

class Solution:

  def tictactoe(self, moves: List[List[int]]) -> str:

    row = [[0] \* 3 for \_ in range(2)]

    col = [[0] \* 3 for \_ in range(2)]

    dg1 = [0] \* 2

    dg2 = [0] \* 2

    i = 0

    for r, c in moves:

      row[i][r] += 1

      col[i][c] += 1

      dg1[i] += r == c

      dg2[i] += r + c == 2

      if 3 in (row[i][r], col[i][c], dg1[i], dg2[i]):

        return "A" if i == 0 else "B"

      i ^= 1

    return "Draw" if len(moves) == 9 else "Pending"

**1281 Subtract the Product and Sum of Digits of an Integer**

class Solution:

  def subtractProductAndSum(self, n: int) -> int:

    prod = 1

    summ = 0

    while n > 0:

      prod \*= n % 10

      summ += n % 10

      n //= 10

    return prod - summ

**1290 Convert Binary Number in a Linked List to Integer**

class Solution:

  def getDecimalValue(self, head: ListNode) -> int:

    ans = 0

    while head:

      ans = ans \* 2 + head.val

      head = head.next

    return ans

**1295 Find Numbers with Even Number of Digits**

class Solution:

    def findNumbers(self, nums: List[int]) -> int:

        ans = 0

        for i in range(len(nums)):

            if len(str(nums[i])) % 2 == 0:

                ans += 1

        return ans

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**