**Facebook 一亩三分地面经**

2 coding interviews

1 system interviews

1 PhD / BQ interview

6 weeks bootcamp

Decision 2 weeks after onsite interviews

**给两个sorted integer array，找出来所有相同的integer**

Two pointers

**给一个long string，找出来所有的palindrome combi‍‍‍‍‍‌‍‍‌‌‍‍‍‍‍‌‍nation**

**LeetCode 415 Add Strings**

class Solution:

def addStrings(self, num1, num2):

"""

:type num1: str

:type num2: str

:rtype: str

"""

i = 1

carry = 0

result = ''

while i <= len(num1) or i <= len(num2):

temp = carry

if i <= len(num1):

temp += ord(num1[-i]) - ord('0')

if i <= len(num2):

temp += ord(num2[-i]) - ord('0')

if temp >= 10:

temp -= 10

carry = 1

else:

carry = 0

# print(i, temp, carry)

result = str(temp) + result

i += 1

if carry > 0:

result = str(carry) + result

return result

**String Rotate & rotate substring from i to j**

**Goat Latin Language**

**LeetCode 236 Lowest Common Ancestor of a Binary Tree**

def lowest\_ancestor(root, A, B):

if root is None:

return root

if root is A or root is B:

return root

left = self.lowest\_ancestor(root.left, A, B)

right = self.lowest\_ancestor(root.right, A, B)

if left is not None and right is not None:

return root

if left is None:

return right

if right is None:

return left

return root

**LeetCode 438 Find All Anagrams in a String**

**LeetCode 49 Group Anagrams**

**LeetCode 224 Basic Calculator**

def calculate(s):

i = 0

stack = []

result = 0

sign = 1

while i < len(s):

char = s[i]

if char == ‘+’:

sign = 1

i += 1

elif char == ‘-’:

sign = -1

i += 1

elif char == ‘(‘:

stack.append((sign, result))

sign = 1

result = 0

i += 1

elif char == ‘)’:

prev\_sign, prev\_result = stack.pop()

result \*= prev\_sign

result += prev\_result

i += 1

elif char == ‘ ‘:

i += 1

else:

temp = 0

while i < len(s) and s[i] not in set([‘(‘, ‘)’, ‘ ‘, ‘+’, ‘-’]):

temp = temp \* 10 + int(s[i])

i += 1

result += sign \* temp

return result

**Subarray Sum Equals K**

def subarra\_sum(nums, k):

sums = dict()

sums[0] = 1

s = 0

result = 0

for num in nums:

s += num

result += sums.get(s-k, 0)

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

return result

**LeetCode 269 Alien Dictionary**

Example 1:

Input:

[

"wrt",

"wrf",

"er",

"ett",

"rftt"

]

Output: "wertf"

Example 2:

Input:

[

"z",

"x"

]

Output: "zx"

**Topology Traversal**

**Meeting Rooms II**

import heapq

def min\_meeting\_rooms(intervals):

if intervals is None or len(intervals) == 0:

return 0

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

heap = []

for interval in intervals:

if len(heap) > 0 and interval[0] >= heap[0]:

heapq.heapreplace(heap, interval[1])

else:

heapq.heappush(heap, interval[1])

return len(heap)

**LeetCode 621 Task Scheduler**

Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

Explanation: A -> B -> idle -> A -> B -> idle -> A -> B.

def least\_interval(tasks, n):

counts\_d = dict()

for t in tasks:

counts\_d[t] = counts\_d.get(t, 0) + 1

counts = [counts\_d[t] for t in counts\_d]

counts.sort(reverse=True)

result = 0

while counts.[0] > 0:

i = 0

for i in range(n):

if counts[0] == 0:

break

if i < len(counts) and counts[i] > 0:

counts[i] -= 1

result += 1

counts.sort(reverse=True)

return result

**LeetCode 938 Range Sum of BST**

def range\_sum\_bst(root, l, r):

def dfs(node):

if node is None:

return

if node.val >= l and node.val <= r:

self.ans += node.val

if node.val >= l

dfs(node.left)

if node.val <= r:

dfs(node.right)

self.ans = 0

dfs(root)

return self.ans

**给数组 A = [1, 2, [3, [4], 5]]，嵌套list，求和。每层乘以level，level根据嵌套层递增。**

**example 1\*{1 + 2 + 2\*[3 + 3\*(4) ‍‍‍‍‍‌‍‍‌‌‍‍‍‍‍‌‍+ 5]**

def nested\_procuct(level, A):

result = 0

for item in A:

if type(item) == int:

result += item

if type(item) == list:

result += nested\_procuct(level+1, item)

return level \* result

**LeetCode 387 First Unique Character in a String**

**LeetCode 51 N-queens**

def solve\_n\_queens(n):

results = []

xy\_sum = set()

xy\_diff = set()

self.dfs(n, [], results, xy\_sum, xy\_diff)

return results

def dfs(self, n, path, results):

if len(path) == n:

results.append(path[:])

return

p = len(path)

for q in range(n):

if q in path or p + q in xy\_sum or p - q in xy\_diff:

continue

self.dfs(n, path + [q], results, xy\_sum.union([p+q]), xy\_diff.union([p-q]))

**LeetCode 79 Word Search**

def exist(board, word):

for i in range(len(board)):

for j in range(len(board)):

if helper(board, word, i, j):

return True

return False

def helper(board, word, i, j):

if len(word) == 0:

return True

if i < 0 or j < 0 or i >= len(word) or j >= len(word):

return False

if board[i][j] != word[0]:

return False

save = board[i][j]

board[i][j] = ‘.’

result = (

helper(board, word[1:], i-1, j),

helper(board, word[1:], i+1, j),

helper(board, word[1:], i, j-1),

helper(board, word[1:], i, j+1)

)

board[i][j] = save

return result

**LeetCode 126 Word Ladder II**

**LeetCode 301 Remove Invalid Parentheses**

def remove\_invalid\_parenthese(s):

results = []

visited = set()

self.dfs(s, results, visited)

return results

def dfs(self, s, results, visited):

mi = self.calc(s)

if mi == 0:

results.append(s)

return

for i in range(len(s)):

if s[i] not in set([‘(’, ‘)’]):

continue

ns = s[:i] + s[i+1:]

if ns not in visited and self.calc(ns) < mi:

visited.add(ns)

self.dfs(ns, results, visited)

def calc(self, s):

l = 0

r = 0

for c in s:

if c not in set([‘(‘, ‘)’]):

continue

if c == ‘(‘:

l += 1

elif c == ‘)’:

l -= 1

if l < 0:

r += 1

l = 0

return l + r

**LeetCode 129 Sum Root to Leaf Numbers**

def sum\_numbers(self, root):

if root is None:

return 0

results = []

self.dfs(root, [], results)

result = 0

for path in results:

temp = 0

for num in path:

temp = temp \* 10 + num

result += temp

return result

def dfs(self, root, path, results):

if root is None:

return

if root.left is None and root.right is None:

results.append(path[:] + [root.val])

return

self.dfs(root.left, path + [root.val], results)

self.dfs(root.right, path + [root.val], results)

**LeetCode 1047 Remove All Adjacent Duplicates in String**

def remove\_duplicates(S):

stack = []

for c in stack:

if len(stack) > 0 and stack[-1] == c:

stack.pop()

else:

stack.append(c)

return ‘’.join(stack)

**LeetCode 34 Find First and Last Position of Element in Sorted Array**

def search\_range(nums, target):

if nums is None or len(nums) == 0:

return [-1, -1]

results =[]

n = len(nums)

l = 0

r = n - 1

while l + 1 < r:

m = (l + r) // 2

if nums[m] >= target:

r = m

else:

l = m + 1

if nums[l] == target:

results.append(l)

elif nums[r] == target:

results.append(r)

else:

return [-1, -1]

l = 0

r = n - 1

while l + 1 < r:

m = (l + r) // 2

if nums[m] <= target:

l = m

else:

r = m - 1

if nums[r] == target:

results.append(r)

elif nums[l] == target:

results.append(l)

else:

return [-1, -1]

return results

**类似file system，没在地里见过这道题，可能是lz太懒了没怎么刷面经，面试的时候理解题目花了一小会。**

**input = [/a/b/c,/a/b,/x/y]**

**output = [/a/b,/x/y]**

**input = [/a/b/c,/a/c]**

**output = [/a/b/c,a/c]**

**input是list of file path，如果这个path的prefix和input里的某一个file‍‍‍‍‍‌‍‍‌‌‍‍‍‍‍‌‍ path一样，就不需要把这个path放进output里。顺序是unordered.**

Use trie

**LeetCode 278 First bad version**

Binary Search

**LeetCode 153 Find Minimum in Rotated Sorted Array**

def find\_min(nums):

if nums is None or len(nums) == 0:

return 0

l = 0

r = len(nums) - 1

while l + 1 < r:

m = (l + r) // 2

if nums[m] > nums[r]:

l = m

else:

r = m

return min(nums[m], nums[r])

**LeetCode 721 Account Merge**

Union-Find

**LeetCode 664 Strange Printer**

**There is a strange printer with the following two special requirements:**

**The printer can only print a sequence of the same character each time.**

**At each turn, the printer can print new characters starting from and ending at any places, and will cover the original existing characters.**

**Given a string consists of lower English letters only, your job is to count the minimum number of turns the printer needed in order to print it.**

**Example 1:**

**Input: "aaabbb"**

**Output: 2**

**Explanation: Print "aaa" first and then print "bbb".**

**Example 2:**

**Input: "aba"**

**Output: 2**

**Explanation: Print "aaa" first and then print "b" from the second place of the string, which will cover the existing character 'a'.**

**Hint: Length of the given string will not exceed 100.**

dp[i][j] stands for the minimal turns we need for string from index i to index j

dp[i][i] = 1

dp[i][i+1] = 1 if s[i] == s[i+1]

dp[i][i+1] = 2 if s[i] != s[i+1]

The maximal value for dp[start][start+len] is len + 1 : print one char each time

Divide the substring into 2 parts start -> start + k & start + k + 1 -> len

def strange\_printer(s):

if s is None or len(s) == 0:

return 0

n = len(s)

dp = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(i, n):

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

for i in reversed(range(n)):

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

if j == i + 1:

dp[i][j] = 1 if s[i] == s[j] else 2

continue

for k in range(i, j):

dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j])

if s[i] == s[j]:

dp[i][j] -= 1

return dp[0][n-1]

**LeetCode 694 Number of Distinct Islands**

def num\_distinct\_islands(grid):

seen = set()

def explore(r, c, r0, c0):

if r < 0 or r >= len(grid) or c < 0 or c >= len(grid[0])) or grid[r][c] == 0 or (r, c) in seen:

return

seen.add((r, c))

shape.add((r-r0, c-c0))

explore(r-1, c, r0, c0)

explore(r+1, c, r0, c0)

explore(r, c-1, r0, c0)

explore(r, c+1, r0, c0)

shapes = set()

for r in range(len(grid)):

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

shape = set()

explore(r, c, r, c)

if len(shape) > 0:

shapes.add(frozenset(shape))

return len(shapes)

**LeetCode 135 Candy**

def candy(self, ratings: List[int]) -> int:

if ratings is None or len(ratings) == 0:

return 0

n = len(ratings)

candies = [1] \* n

for i in range(1, n):

if ratings[i] > ratings[i-1]:

candies[i] = candies[i-1] + 1

# print(candies)

for i in range(n-2, -1, -1):

if ratings[i] > ratings[i+1]:

candies[i] = max(candies[i], candies[i+1] + 1)

# print(candies)

return sum(candies)

**LeetCode 263 Ugly Number**

def is\_ugly(num):

if num <= 0:

return False

if num in set([1, 2, 3, 4, 5]):

return True

if num % 2 == 0:

return is\_ugly(num // 2)

elif num % 3 == 0:

return is\_ugly(num //3)

elif num % 5 == 0:

return is\_ugly(num//5)

return False

**LeetCode 211: Add and Search Word**

def Node:

def \_\_init\_\_(self):

self.children = dict()

self.is\_word = False

class WordDictionary:

def \_\_init\_\_(self):

self.root = Node()

def add\_word(self, word):

cur = self.root

for ch in word:

if ch not in cur.children:

cur.children[ch] = Node()

cur = cur.children[ch]

cur.is\_word = True

def search(self, word):

return self.helper(word, self.root)

def helper(self, word, node):

for char in word:

if char == ‘.’:

for ch in node.children:

if self.helper(word[1:], node.children[ch]:

return True

return False

else:

if char not in node.children:

return False

return self.helper(word[1:], node.children[char])

return node.is\_word

**273 Integers to English Words**

def \_\_init\_\_(self):

self.twenties = ['', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine', 'Ten', 'Eleven', 'Twelve', 'Thirteen', 'Fourteen', 'Fifteen', 'Sixteen', 'Seventeen', 'Eighteen', 'Nineteen']

self.tens = ['', 'Ten', 'Twenty', 'Thirty', 'Forty', 'Fifty', 'Sixty', 'Seventy', 'Eighty', 'Ninety']

self.thousands = ['', 'Thousand', 'Million', 'Billion']

def numberToWords(self, num: int) -> str:

if num == 0:

return 'Zero'

result = ''

for i in range(len(self.thousands)):

if num % 1000 != 0:

result = self.helper(num % 1000) + self.thousands[i] + ' ' + result

num //= 1000

return result.strip()

def helper(self, num):

if num == 0:

return ''

elif num < 20:

return self.twenties[num] + ' '

elif num < 100:

return self.tens[num // 10] + ' ' + self.helper(num % 10)

else:

return self.twenties[num // 100] + ' Hundred ' + self.helper(num % 100)

**LeetCode 973 K Closest Points to Origin**

class Solution:

def kClosest(self, points, K):

if points is None or len(points) == 0:

return points

n = len(points)

dist = lamda i: pow(points[i][0], 2) + pow(points[i][1], 2)

self.selection(points, dist, 0, n-1, K)

return points[:K]

def select(self, points, dist, l, r, K):

if l >= r:

return

mid = self.partition(points, dist, l, r)

if mid == K:

return

if mid > K:

self.select(points, dist, l, mid-1, K)

if mid < K:

self.select(points, dist, mid+1, r, K)

def partition(self, points, dist, l , r):

start = l

pivot = dist(r)

for i in range(l, r):

if dist(i) <= pivot:

points[start], points[i] = points[i], points[start]

start += 1

points[start], points[r] = points[r], points[start]

return start

**LeetCode 300 Longest Increasing Subsequence**

dp: dynamic programming

def length\_of\_LIS(nums):

if nums is None or len(nums) == 0:

return 0

dp = [1] \* len(nums)

for i in range(len(nums)):

for j in range(i):

if nums[j] < nums[i]:

dp[i] = max(dp[i], dp[j] + 1)

return max(dp)

def length\_of\_LIS(nums):

n = len(nums)

ups = []

for i in range(n):

l = 0

r = len(ups) -

while l + 1 < r:

m = (l + r) // 2

if ups[m] <= nums[i]:

l = m

else:

r = m

if l >= len(ups):

ups.append(nums[i])

elif ups[l] >= nums[i]:

ups[l] = nums[i]

elif r >= len(ups):

ups.append(nums[i])

elif ups[r] >= nums[i]:

ups[r] = nums[i]

else:

ups.append(nums[i])

return len(ups)

**Dynamic Programming with Binary Search**

def length\_of\_LIS(nums):

dp = []

len = 0

n = len(nums)

for i in range(n):

j = binary\_search(dp, nums[i])

if j < 0:

j = -(j + 1)

if j > len:

dp.append(nums[j])

len += 1

else:

dp[j] = nums[i]

return len

**LeetCode 678 Valid Parenthesis**

def check\_valid\_string(self, s):

if s is None or len(s) == 0:

return False

cmin = 0

cmax = 0

for c in s:

if c == ‘(‘:

cmin += 1

cmax += 1

if c == ‘)’:

cmax -= 1

cmin = max(cmin - 1, 0)

if c == ‘\*’:

cmax += 1

cmin = max(cmin-1, 0)

if cmax < 0:

return False

return cmin == 0

**LeetCode 91 Decode Ways**

def num\_decoding(s):

if s is None or len(s) == 0:

return 0

dp = [0] \* (len(s) + 1)

dp[0] = 1

for i in range(len(s)):

if int(s[i]) >= 1:

dp[i+1] += dp[i]

if i >= 1 and int(s[i-1:i+1]) >= 10 and int(s[i-1:i+1]) <= 26:

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

if dp[i+1] == 0:

return 0

return dp[-1]

**LeetCode 639 Decode Ways II**

def num\_decodings(s):

mod = 10 \*\* 9 + 7

# e0 decode in any endings

# e1 decode ended with 1

# e2 decode ended with 2

e0, e1, e2 = 1, 0, 0

for c in s:

if c != '\*':

f0 = (c > '0') \* e0 + e1 + (c<= '6') \* e2

f1 = (c == '1') \* e0

f2 = (c == '2') \* e0

else:

f0 = 9 \* e0 + 9 \* e1 + 6 \* e2

f1 = e0

f2 = e0

# print(e0, e1, e2, f0, f1, f2)

e0, e1, e2 = f0 % mod, f1 % mod, f2 % mod

return e0

**LeetCode 692 Top K Frequent Words**

import collections

import heapq

import functools

@functools.total\_ordering

class Element:

def \_\_init\_\_(self, count, word):

self.count = count

self.word = word

def \_\_lt\_\_(self, other):

if self.count == other.count:

return self.word > other.word

return self.count < other.count

def \_\_eq\_\_(self, other):

return self.count == other.count and self.word == other.word

class Solution:

def top\_k\_frequent(self, words, k):

counts = collections.Counter(words)

freqs = []

heapq.heapify(freqs)

for word, count in counts.items():

heapq.heappush(freqs, (Element(count, word), word))

if len(freqs) > k:

heapq.heappop(freqs)

res = []

for \_ in range(k):

res.append(heapq.heappop(freqs)[1])

return reversed(res)

**LeetCode 419 Battleship in a Board**

class Solution:

def countBattleships(self, board: List[List[str]]) -> int:

if board is None or len(board) == 0 or len(board[0]) == 0:

return 0

count = 0

m = len(board)

n = len(board[0])

for i in range(m):

for j in range(n):

if board[i][j] == 'X' and (i == 0 or board[i-1][j] != 'X') and (j == 0 or board[i][j-1] != 'X'):

count += 1

return count

**LeetCode 238 Product of Array Except Self**

left product list & right product list

To use constant space

def product\_except\_self(nums):

n = len(nums)

answer = [0] \* n

answer[0] = 1

for i in range(1, n):

answer[i] = answer[i-1] \* nums[i-1]

R = 1

for i in reversed(range(n)):

answer[i] \*= R

R \*= nums[i]

return answer

**LeetCode 23 Merge K sorted lists**

import heapq

def merge\_k\_lists(lists):

if lists is None or len(lists) == 0:

return None

heap = []

heapq.heapify(heap)

for node in lists:

while node is not None:

heap.heappush(heap, node.val)

node = node.next

head = ListNode(-1)

cur = head

while len(heap) > 0:

cur.next = ListNode(heapq.heappop(heap))

cur = cur.next

return head.next

**LeetCode 42: Trapping Rain Water**

def trap(heights):

if height is None or len(heights) == 0:

return 0

n = len(heights)

l = 0

r = n - 1

min\_height = 0

result = 0

while l < r:

while l < r and height[l] <= min\_height:

result += min\_height - height[l]

l += 1

while l < r and height[r] <= min\_height:

result += min\_height - height[r]

r -= 1

min\_height = min(height[l], height[r])

return result

**LeetCode 958: Check completeness of a binary tree**

def is\_complete(root):

nodes = [(root, 1)]

i = 0

while i < len(nodes):

node, v = nodes[i]

i += 1

if node:

nodes.append((node.left, v \* 2))

nodes.append((node.right, v \* 2 + 1))

return nodes[-1][1] == len(nodes)

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

seq = []

queue = [(root, 1)]

while len(queue) > 0:

node, v = queue.pop(0)

seq.append(v)

if node.left:

queue.append((node.left, v \* 2))

if node.right:

queue.append((node.right, v \* 2 + 1))

n = len(seq)

for i in range(n-1):

if seq[i] != seq[i+1] - 1:

return False

return True

**计算Binary Tree所有叶子节点到根的路径之和**

**Intersection of 2 Arrays**

o(n) time & o(1) space

2 pointers

**LintCode 183 Wood Cut**

Description

Given n pieces of wood with length L[i] (integer array). Cut them into small pieces to guarantee you could have equal or more than k pieces with the same length. What is the longest length you can get from the n pieces of wood? Given L & k, return the maximum length of the small pieces.

**Binary search**

**有两个List，分别代表两个人Calendar上面的空闲时间段。比如：**

**A = [[1, 3], [5, 7]‍‍‍‍‍‌‍‍‌‌‍‍‍‍‍‌‌]**

**B = [[2, 4], [6, 8]]**

**要求输出两人共同的空闲时间段。这题返回[[2,3], [6,7]]就好**

**LeetCode 146 LRU cache**

class Node:

def \_\_init\_\_(self, key, val, prev=None, next=None):

self.key = key

self.val = val

self.prev = prev

self.next = next

class LRUCache:

def \_\_init\_\_(self, capacity):

self.capacity = capacity

self.dict = dict()

self.head = Node(0, 0)

self.tail = Node(0, 0)

def get(self, key):

if key in self.dict:

n = self.dict[key]

self.\_remove(n)

self.\_add(n)

return n.val

return -1

def put(self, key, value):

if key in self.dict:

n = self.dict[key]

self.\_remove(n)

n = Node(key, value)

self.dict[key] = n

self.\_add(n)

if len(self.dict) > self.capacity:

p = self.head.next

self.\_remove(p)

del self.dict[p.key]

def \_add(self, n):

p = self.tail.prev

p.next = n

n.next = self.tail

n.prev = p

self.tail.prev = n

self.\_remove(self, n):

prev = n.prev

nex = n.next

prev.next = nex

nex.prev = prev

**LeetCode 680. Valid Palindrome II**

delete at most one character, and check whether if it’s valid palindrome

def valid\_palindrom(s):

i = 0

while i < len(s) // 2 and s[i] == s[-(i+1)]:

i += 1

s = s[i: len(s) - i]

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

**给两个sorted integer array，找出来所有相同的integer**

**LeetCode 73. Set Matrix Zeros**

1. We iterate over the matrix and we mark the first cell of a row i and first cell of a column j, if the condition in the pseudo code above is satisfied. i.e. if cell[i][j] == 0.
2. The first cell of row and column for the first row and first column is the same i.e. cell[0][0]. Hence, we use an additional variable to tell us if the first column had been marked or not and the cell[0][0] would be used to tell the same for the first row.
3. Now, we iterate over the original matrix starting from second row and second column i.e. matrix[1][1] onwards. For every cell we check if the row r or column c had been marked earlier by checking the respective first row cell or first column cell. If any of them was marked, we set the value in the cell to 0. Note the first row and first column serve as the row\_set and column\_set that we used in the first approach.
4. We then check if cell[0][0] == 0, if this is the case, we mark the first row as zero.
5. And finally, we check if the first column was marked, we make all entries in it as zeros.

def set\_zeros(matrix):

is\_col = False

is\_row = False

r = len(matrix)

c = len(matrix[0])

for i in range(r):

if matrix[i][0] == 0:

is\_col = True

for j in range(c):

if i == 0 and matrix[i][j] == 0:

is\_row = True

if matrix[i][j] == 0:

matrix[0][j] = 0

matrix[i][0] = 0

for i in range(1, r):

for j in range(1, c):

if matrix[i][0] == 0 or matrix[0][j] == 0:

matrix[i][j] = 0

if is\_col:

for i in range(r):

matrix[i][0] = 0

if is\_row:

for j in range(c):

matrix[0][j] = 0

**LeetCode 236 Lowest Common Ancestor Tree**

**LeetCode 438 Find All Anagrams in a String**

Sliding window + Counter

**LeetCode 88 Merge Two Sorted Arrays**

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

The number of elements initialized in nums1 and nums2 are m and n respectively.

You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2.

def merge(nums1, m, nums2, n):

while m > 0 and n > 0:

if nums1[m] >= nums2[n]:

nums1[m+n-1] = nums1[m-1]

m -= 1

else:

nums1[m+n-1] = nums2[n-1]

if n > 0:

nums1[:n] = nums2[:n]

**LeetCode 567 Permutation in String**

**LeetCode 3 Longest Substring Without Repeating Characters**

def length\_of\_longest\_substring(s):

if s is None or len(s) == 0:

return 0

n = len(s)

res = 1

l = -1

mem = dict()

for i in range(n):

if s[i] in mem:

l = max(l, mem[s[i]])

mem[s[i]] = i

res = max(res, i - l)

return res

**LeetCode 242 Valid Anagram**

**LeetCode 498 Diagonal Traverse**

class Solution:

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

if matrix is None or len(matrix) == 0:

return []

m = len(matrix)

n = len(matrix[0])

results = []

r = 0

c = 0

for i in range(m \* n):

print(r, c)

results.append(matrix[r][c])

if (r + c) % 2 == 0:

if c == n - 1:

r += 1

elif r == 0:

c += 1

else:

r -= 1

c += 1

else:

if r == m - 1:

c += 1

elif c == 0:

r += 1

else:

r += 1

c -= 1

return results

**LeetCode 50 Pow(x, n)**

def my\_pow(x, n):

if n == 0:

return 1

if n == 1:

return x

flag = False

if n < 0:

flag = True

n = -n

temp = self.my\_pow(x, n // 2)

if n % 2 == 0:

result = temp \* temp

else:

result = temp \* temp \* x

if flag:

return 1 / result

return result

**LeetCode 1123 Lowest Common Ancestor of Deepest Leaves**

def lcaDeepestLeaves(self, root: TreeNode) -> TreeNode:

def helper(node):

if node is None:

return 0, None

h1, lca1 = helper(node.left)

h2, lca2 = helper(node.right)

if h1 > h2:

return h1 + 1, lca1

if h1 < h2:

return h2 + 1, lca2

return h1 + 1, node

return helper(root)[1]

**LeetCode 261 Graph Valid Tree**

class Solution:

def validTree(self, n: int, edges: List[List[int]]) -> bool:

if len(edges) != n - 1:

return False

roots = dict()

for i in range(n):

roots[i] = i

for edge in edges:

root1 = self.find(roots, edge[0])

root2 = self.find(roots, edge[1])

if root1 == root2:

return False

self.union(roots, edge[0], edge[1])

return True

def find(self, roots, i):

if roots[i] != i:

roots[i] = self.find(roots, roots[i])

return roots[i]

def union(self, roots, i, j):

root\_i = self.find(roots, i)

root\_j = self.find(roots, j)

if root\_i != root\_j:

roots[root\_i] = root\_j

**LeetCode 426 Convert Binary Search Tree to a Sorted Doubly Linked List**

class Solution:

def treeToDoublyList(self, root: 'Node') -> 'Node':

if root is None:

return None

head = Node(0, None, None)

self.prev = head

self.helper(root)

self.prev.right = head.right

head.right.left = self.prev

return head.right

def helper(self, node):

if node is None:

return

self.helper(node.left)

self.prev.right = node

node.left = self.prev

self.prev = node

self.helper(node.right)