**Facebook 面试频率最高100题**

**LeetCode 56. Merge Intervals**

class Solution

def merge\_intervals(intervals):

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

return []

results = []

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

for interval in intervals:

if len(results) == 0 or interval[0] > results[-1][1]:

results.append(interval)

else:

results[-1][1] = max(results[-1][1], interval[1])

return results

**LeetCode 252. Meeting Rooms**

class Solution:

def can\_attend\_meetings(intervals):

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

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

if intervals[i][1] > intervals[i+1][0]:

return False

return True

**LeetCode 253. Meeting Rooms II**

import heapq

class Solution:

def min\_meeting\_rooms(intervals):

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

return 0

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

rooms = []

for inv in intervals:

if len(rooms) > 0 and inv[0] >= rooms[0]:

heapq.heappop(rooms)

heapq.heappush(rooms, inv[1])

else:

heapq.heappush(rooms, inv[1])

return len(rooms)

**LeetCode 15. 3Sum**

class Solution:

def three\_sum(self, nums):

if nums is None or len(nums) <= 2:

return []

nums.sort()

results = []

n = len(nums)

for i in range(n-2):

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

continue

l = i + 1

r = n - 1

while l < r:

s = nums[i] + nums[l] + nums[r]

if s < 0:

l += 1

elif s > 0:

r -= 1

else:

results.append((nums[i], nums[l], nums[r]))

while l < r and nums[l] == nums[l+1]:

l += 1

while l < r and nums[r] == nums[r-1]:

r -= 1

l += 1

r -= 1

return results

**LeetCode 75. Sort Colors**

class Solution:

def sort\_colors(nums):

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

return

red = 0

white = 0

blue = len(nums) - 1

# red: [red, white]

# white: [white+1, blue]

# blue: [blue+1:]

while whilte < blue:

if nums[white] == 0:

nums[red], nums[white] = nums[white], nums[red]

red += 1

white += 1

elif nums[white] == 1:

white += 1

else:

nums[white], nums[blue] = nums[blue], nums[white]

blue -= 1

return

**LeetCode 825. Friends of Appropriate Ages**

Person A will NOT friend request person B (B != A) if any of the following conditions are true:

age[B] <= 0.5 \* age[A] + 7

age[B] > age[A]

age[B] > 100 && age[A] < 100

Otherwise, A will friend request B.

import collections

class Solution:

def num\_friend\_requests(self, ages):

result = 0

counts = collections.Counter(ages)

for a in counts:

for b in counts:

if b <= 0.5 \* a + 7 or b > a:

continue

results += counts[a] \* (counts[b] - (a==b))

return result

**LeetCode 621. Task Scheduler**

import collections

class Solution:

def least\_interval(self, tasks, n):

counts\_d = collections.Counter(tasks)

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

counts.sort(reverse = True)

result = 0

while counts[0] > 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 863. All Nodes Distance K in Binary Tree**

class Solution:

def distanceK(self, root, target, K):

def dfs(node, par=None):

if node is None:

return

node.par = par

dfs(node.left, node)

dfs(node.right, node)

dfs(root, None)

queue = [(target, 0)]

seen = set([target])

while queue:

if queue[0][1] == K:

return [node.val for node, d in queue]

node, d = queue.pop(0)

for nei in (node.left, node.right, node.par):

if nei is not None and nei not in seen:

seen.add(nei)

queue.append((nei, d+1))

return []

**LeetCode 53. Maximum Subarray**

class Solution:

def max\_subarray(self, nums):

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

return 0

result = - float(‘inf’)

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

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

dp[i] = nums[i-1] + max(dp[i-1], 0)

result = max(result, dp[i])

return result

**LeetCode 88. Merge Sorted Array**

class Solution:

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

while m > 0 and n > 0:

if nums[m-1] >= nums2[n-1]:

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

else:

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

if n > 0:

nums1[:n] = nums2[:n]

**LeetCode 27. Remove Element**

class Solution:

def remove\_element(self, nums, val):

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

return 0

tail = len(nums) - 1

i = 0

while i <= tail:

if nums[i] == val:

nums[i], nums[tail] = nums[tail], nums[i]

tail -= 1

else:

i += 1

return i

**LeetCode 674. Longest Continuous Increasing Subsequence**

class Solution:

def find\_length\_of\_LCIS(self, nums):

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

return 0

temp = 1

result = 1

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

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

temp += 1

else:

temp = 1

result = max(result, temp)

return result

**LeetCode 283. Move Zeros**

class Solution:

def move\_zeros(self, nums):

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

return

tail = 0

i = 0

while i < len(nums):

while tail < len(nums) and nums[tail] != 0:

tail += 1

i = tail

while i < len(nums) and nums[i] == 0:

i += 1

if i < len(nums) and tail < len(nums):

nums[i], nums[tail] = nums[tail], nums[i]

tail += 1

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

import collections

class Solution:

def find\_anagrams(self, s, p):

results = []

\_p = collections.Counter([c for c in p])

\_s = collections.Counter([c for c in s[:len(p)]])

for i in range(len(s) - len(p) + 1):

if i > 0:

\_s[s[i-1]] -= 1

\_s[s[i+len(p)-1] += 1

if self.check(\_s, \_p):

results.append(i)

return results

def check(self, \_s, \_p):

for c in \_s:

if \_s[c] != 0 and (c not in \_p or \_s[c] != \_p[c]):

return False

return True

**LintCode 767. Reverse Array**

class Solution:

def reverse\_array(self, nums):  
 if nums is None or len(nums) == 0:

return

i = 0

j = len(nums) - 1

while i < j:

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

i += 1

j -= 1

**LeetCode 896. Monotonic Array**

class Solution:

def is\_monotonic(self, A):

increasing = True

decreasing = True

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

if A[i] < A[i+1]:

decreasing = False

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

increasing = False

return increasing or decreasing

**LeetCode 31. Next Permutation**

class Solution:

def next\_permutation(self, nums):

if nums is None or len(nums) <= 1:

return

n = len(nums)

i = n - 1

while i >= 1 and nums[i-1] >= nums[i]:

i -= 1

pivot = i - 1

next\_pivot = pivot + 1

for i in range(next\_pivot, n):

if nums[i] <= nums[pivot]:

continue

if nums[i] < nums[next\_pivot]:

next\_pivot = i

nums[pivot], nums[next\_pivot] = nums[next\_pivot], nums[pivot]

nums[pivot+1:] = sorted(nums[pivot+1:])

**LeetCode 33. Search in Rotated Sorted Array**

class Solution:

def search(self, nums, target):

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

return -1

l = 0

r = len(nums-1)

while l + 1 < r:

m = (l + r) // 2

if nums[m] == target:

return m

if nums[m] > nums[l]:

if nums[l] <= target <= nums[m]:

r = m

else:

l = m

else:

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

l = m

else:

r = m

if nums[l] == target:

return l

if nums[r] == target:

return r

return -1

**LeetCode 121. Best Time to Buy and Sell Stock**

class Solution:

def max\_profit(self, prices):

min\_price = float(‘inf’)

max\_profit = 0

for i in range(len(prices)):

if prices[i] < min\_price:

min\_price = prices[i]

elif prices[i] - min\_price > max\_profit:

max\_profit = prices[i] - min\_price

return max\_profit

**LeetCode 42. Trapping Rain Water**

class Solution:

def trap(self, height):

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

return 0

l = 0

r = len(height) - 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 523. Continuous Subarray Sum**

class Solution:

def check\_subarray\_sum(nums, k):

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

return False

n = len(nums)

for i in range(n-1):

if nums[i] == 0 and nums[i+1] == 0:

return True

if k == 0: return False

if k < 0: k = -k

sum2index = dict()

sum2index[0] = -1

sum = 0

for i in range(len(nums)):

sum += nums[i]

for j in range(k, sum, k):

if (sum - j) in sum2index and i - sum2index[sum - j] >= 2:

return True

if sum not in sum2index: sum2index[sum] = i

return False

**LeetCode 135. Candy**

class Solution:

def candy(self, ratings):

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

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

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

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

return sum(candies)

**LeetCode 380. Insert Delete GetRandom O(1)**

import random

class RandomizedSet:

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

self.set = list()

self.index = dict()

def insert(self, val):

if val not in self.index:

self.set.append(val)

self.index[val] = len(self.set) - 1

return True

return False

def remove(self.val):

if val in self.index:

ind = self.index[val]

self.set[ind] = self.set[-1]

self.index[self.set[ind]] = ind

self.set.pop()

self.index.pop(val)

return True

return False

def getRandom(self):

return random.choice(self.set))

**LeetCode 621. Task Scheduler**

import collections

class Solution:

def least\_interval(self, tasks, n):

counts\_d = collections.Counter([t for t in tasks])

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

counts.sort(reverse=True)

result = 0

while counts[0] > 0:

for i in range(n+1):

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 714. Best Time to Buy and Sell Stock with Transaction Fee**

At the end of the i-th day, we maintain cash, the maximum profit we could have if we did not have a share of stock, and hold, the maximum profit we could have if we owned a share of stock.

To transition from the i-th day to the i+1-th day, we either sell our stock cash = max(cash, hold + prices[i] - fee) or buy a stock hold = max(hold, cash - prices[i]). At the end, we want to return cash. We can transform cash first without using temporary variables because selling and buying on the same day can't be better than just continuing to hold the stock.

class Solution:

def max\_profit(self, prices, fee):

cash = 0

hold = -prices[0]

for i in range(1, len(prices)):

cash = max(cash, hold + prices[i] - fee)

hold = max(hold, cash - prices[i])

return cash

**LeetCode 498. Diagonal Traverse**

class Solution:

def find\_diagonal\_order(self, matrix):

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):

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

if (r + c) % 2 == 0: # moving up

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 238. Product of Array Except Self**

class Solution:

def product\_except\_self(self, nums):

n = len(nums)

before = [0] \* n

after = [0] \* n

before[0] = nums[0]

for i in range(1, n):

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

after[-1] = nums[-1]

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

after[-i] = after[-i+1] \* nums[-i]

results = [0] \* n

for i in range(n):

if i == 0:

results[i] = after[1]

elif i == n-1:

results[i] = before[i-1]

else:

results[i] = before[i-1] \* after[i+1]

return results

**LeetCode 689. Maximum Sum of 3 Non-Overlapping Subarrays**

class Solution:

def max\_sum\_of\_three\_subarrays(self, nums, k):

n = len(nums)

w = []

inv = 0

for i in range(n):

inv += nums[i]

if i >= k:

inv -= nums[i-k]

if i >= k-1:

w.append(inv)

left = [0] \* len(w)

best = 0

for i in range(len(w)):

if w[i] > w[best]:

best = i

left[i] = best

right = [0] \* len(w)

best = len(w) - 1

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

if w[i] >= w[best]:

best = i

right[i] = best

ans = None

for j in range(k, len(w)-k):

i, m = left[j-k], right[j+k]

if ans is None or w[i] + w[j] + w[m] > w[ans[0]] + w[ans[1]] + w[ans[2]]:

ans = (i, j, m)

return ans

**LeetCode 114. Flatten Binary Tree to Linked List**

class Solution:

def flatten(self, root):

if root is None:

return

if root.left is None and root.right is not None:

self.flatten(root.right)

return

if root.left is not None and root.right is None:

self.flatten(root.left)

root.right = root.left

root.left = None

return

left = root.left

right = root.right

self.flatten(left)

self.flatten(right)

root.left = None

root.right = left

now = root

while now.right:

now = now.right

now.right = right

**LeetCode 349. Intersection of Two Arrays**

class Solution:

def intersection(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: List[int]

"""

results = []

for num in nums1:

if num in nums2 and num not in results:

results.append(num)

return results

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

class Solution:

def findAnagrams(self, s, p):

"""

:type s: str

:type p: str

:rtype: List[int]

"""

results = []

\_p = dict()

for c in p:

\_p[c] = \_p.get(c, 0) + 1

\_s = dict()

for c in s[:len(p)]:

\_s[c] = \_s.get(c, 0) + 1

# print(\_p, \_s)

for i in range(len(s) - len(p) + 1):

if i > 0:

# print(i, s[i-1], s[i+len(p)])

\_s[s[i-1]] -= 1

\_s[s[i+len(p)-1]] = \_s.get(s[i+len(p)-1], 0) + 1

if self.helper(\_s, \_p):

results.append(i)

return results

def helper(self, \_s, \_p):

for c in \_s:

if \_s[c] != 0 and (c not in \_p or \_s[c] != \_p[c]):

return False

return True

**LeetCode 560. Subarray Sum Equals K**

class Solution:

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

sums = dict()

sums[0] = 1

result = 0

s = 0

for num in nums:

s += num

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

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

return result

**LeetCode 76. Minimum Window Substring**

class Solution:

def minWindow(self, s: str, t: str) -> str:

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

return ''

t\_d = dict()

for i, c in enumerate(t):

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

flag = False

n = len(s)

l = 0

res = s

s\_d = dict()

for r in range(n):

s\_d[s[r]] = s\_d.get(s[r], 0) + 1

while self.helper(s\_d, t\_d):

flag = True

if r - l + 1 < len(res):

res = s[l:r+1]

s\_d[s[l]] -= 1

l += 1

if not flag:

return ''

return res

def helper(self, s\_d, t\_d):

for c in t\_d:

if c not in s\_d or s\_d[c] < t\_d[c]:

return False

return True

**LeetCode 138 Copy List with Random Pointer**

class Solution:

def copyRandomList(self, head):

dummy = ListNode(-1, None, None)

node2copy = dict()

copy\_cur = dummy

cur = head

while cur is not None:

copy\_cur.next = Node(cur.val, None, None)

node2copy[cur] = copy\_cur.next

cur = cur.next

copy\_cur = copy\_cur.next

copy\_cur = dummy

cur = head

while cur is not None:

if cur.random is not None:

copy\_cur.next.random = node2copy[cur.random]

cur = cur.next

copy\_cur = copy\_cur.next

return dummy.next

**LeetCode 133. Clone Graph**

class Solutions:

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

self.visited = dict()

def clone\_graph(self, node):

if node is None:

return None

if node.label is self.visited:

return self.visited[node.label]

clone = UndirectedGraphNode(node.label)

self.visited[clone.label] = clone

for n in node.neighbors:

clone.neighbors.append(self.clone\_graph(n))

return clone

**LeetCode 3. Longest Substring Without Repeating Characters**

class Solution:

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

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

return 0

n = len(s)

res = 1

l = -1

index = dict()

for r in range(n):

if s[r] in memory:

l = max(l, memory[s[r]])

memory[s[r]] = r

res = max(res, r - l)

return res

**LeetCode 987. Vertical Order Traversal of a Binary Tree**

class Solution:

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

seen = dict()

def dfs(node, x, y):

if node is None:

return

if x not in seen:

seen[x] = dict()

if y not in seen[x]:

seen[x][y] = list()

seen[x][y].append(node)

dfs(node.left, x-1, y+1)

dfs(node.right, x+1, y+1)

dfs(root, 0, 0)

ans = []

for x in sorted(seen):

report = []

for y in sorted(seen[x]):

report.extend(sorted(node.val for node in seen[x][y]))

ans.append(report)

return ans

**LeetCode 694. Number of Distinct Islands**

class Solution:

def num\_distinct\_islands(self, grid):

seen = set()

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

if r < 0 or c < 0 or r >= len(grid) or c >= len(grid) 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 297. Serialize and Deserialize Binary Tree**

class Solution:

def serialize(self, root):

vals = []

def helper(root):

if root is None:

vals.append(‘\*’)

else:

vals.append(str(root.val))

helper(root.left)

helper(root.right)

helper(root)

return ‘,’.join(vals)

def de\_serialize(self, data):

vals = iter(data.split(‘,’))

def helper():

node = vals.next()

if node == ‘\*’:

return None

root = TreeNode(int(node))

root.left = helper()

root.right = helper()

return root

return helper()

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

class Solution:

def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':

if root is None:

return None

if root is p or root is q:

return root

left = self.lowestCommonAncestor(root.left, p, q)

right = self.lowestCommonAncestor(root.right, p, q)

if left is None:

return right

if right is None:

return left

return root

**LeetCode 94. Binary Tree Maximum Path Sum**

class Solution:

def max\_path\_sum(self, root):

self.result = -float(‘inf’)

self.helper(root)

return self.result

def helper(self, root):

if root is None:

return 0, 0, 0

l\_l\_max, l\_r\_max, l\_max = self.helper(root.left)

r\_l\_max, r\_r\_max, r\_max = self.helper(root.right)

l\_max = root.val + max(0, l\_l\_max, l\_r\_max)

r\_max = root.val + max(0, r\_l\_max, r\_r\_max)

max\_val = root.val + max(0, l\_l\_max, l\_r\_max) + max(0, r\_l\_max, r\_r\_max)

self.result = max(self.result, max\_val)

return l\_max, r\_max, max\_val

class Solution:

def max\_path\_sum(self, root):

self.max\_sum = -float(‘inf’)

def max\_gain(node):

if node is None:

return 0

left\_gain = max(0, max\_gain(node.left))

right\_gain = max(0, max\_gain(node.right))

path = node.val + left\_gain + right\_gain

self.max\_sum = max(self.max\_sum, path)

return node.val + max(left\_gain, right\_gain)

max\_gain(root)

return self.max\_sum

**LeetCode 199. Binary Tree Right Side View**

class Solution:

def right\_side\_view(self, root):

if root is None:

return []

matrix = []

queue = [root]

new\_queue = []

temp = []

while len(queue) > 0:

node = queue.pop(0)

temp.append(node)

if node.left:

new\_queue.append(node.left)

if node.right:

new\_queue.append(node.right)

if len(queue) == 0:

queue = new\_queue[:]

matrix.append(temp[:])

new\_queue = []

temp = []

results = []

for temp in matrix:

results.append(temp[-1])

return results

**LeetCode 938. Range Sum of BST**

class Solution:

def range\_sum\_bst(self, root, L, R):

self.ans = 0

def dfs(node):

if node is None:

return

if L <= node.val <= R:

self.ans += root.val

if L < node.val:

dfs(node.left)

if node.val < R:

dfs(node.right)

dfs(root)

return self.ans

**LeetCode 242. Valid Anagram**

class Solution:

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

if s is None or t is None:

return False

if len(s) != len(t):

return False

all\_c = set()

s\_d = dict()

for c in s:

all\_c.add(c)

s\_d[c] = s\_d.get(c, 0) + 1

t\_d = dict()

for c in t:

all\_c.add(c)

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

for c in all\_c:

if c not in t\_d or c not in s\_d or s\_d[c] != t\_d[c]:

return False

return True

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

class Solution(object):

def firstUniqChar(self, s):

"""

:type s: str

:rtype: int

"""

before = set()

repeating = set()

for c in s:

if c in before:

repeating.add(c)

before.add(c)

for i in range(len(s)):

if s[i] not in repeating:

return i

return -1

**LeetCode 67. Add Binary**

class Solution:

def addBinary(self, a, b):

"""

:type a: str

:type b: str

:rtype: str

"""

if a == '0':

return b

if b == '0':

return a

results = [0] \* (max(len(a), len(b)) + 1)

a\_list = [int(x) for x in a]

b\_list = [int(x) for x in b]

i = 1

carry = 0

while i <= len(a\_list) or i <= len(b\_list):

temp = carry

if i <= len(a\_list):

temp += a\_list[-i]

if i <= len(b\_list):

temp += b\_list[-i]

if temp >= 2:

results[-i] = temp % 2

carry = temp // 2

else:

results[-i] = temp

carry = 0

i += 1

if carry == 1:

results[-i] = carry

while results and results[0] == 0:

results.pop(0)

return ''.join([str(x) for x in results])

**LeetCode 139. Word Break**

class Solution:

def word\_break(self, s, word\_dict):

if len(word\_dict) == 0:

return False

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

dp[0] = True

for i in range(1, len(s) + 1):

for j in range(i):

if dp[j] and s[j:i] in word\_dict:

dp[i] = True

break

return dp[-1]

**LeetCode 71. Simplify Path**

class Solution:

def simplify\_path(self, path):

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

return path

if path[-1] == ‘/’:

path = path[:-1]

temp = path.split(‘/’)

results = []

for dir in temp:

if dir == ‘.’ or dir == ‘’:

continue

elif dir == ‘..’:

if len(results) > 0:

results.pop()

else:

results.append(dir)

if len(results) >= 2:

return ‘/’ + ‘/’.join(results)

elif len(results) == 1;

return ‘/’ + results[0]

else:

return ‘/’

**LeetCode 72. Edit Distance**

class Solution:

def min\_distance(self, word1, word2):

if word1 is None or word2 is None:

return 0

m = len(word1)

n = len(word2)

dp = [[0] \* (n+1) for \_ in range(m+1)]

for i in range(m+1):

for j in range(n+1):

if i == 0 and j == 0:

continue

elif i == 0:

dp[i][j] = j

elif j == 0:

dp[i][j] = i

elif word1[i-1] == word2[j-1]:

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

else:

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

return dp[-1][-1]

**LeetCode 43. Multiply Strings**

class Solution:

def multiply(self, num1, num2):

if num1 == ‘0’ or num2 == ‘0’:

return ‘0’

m = len(num1)

n = len(num2)

results = [0] \* (m + n)

for i in reversed(range(m)):

digit1 = ord(num1[1]) - ord(‘0’)

for j in reversed(range(n)):  
 digit2 = ord(num2[j]) - ord(‘0’)

results[i+j+1] += digit1 \* digit2

while results[0] == 0:

results = results[1:]

carry = 0

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

results[i] = results[i] + carry

if results[i] >= 10:

carry = results[i] // 10

results[i] = results[i] % 10

else:

carry = 0

if carry > 0:

results.insert(0, carry)

return ‘’.join([str(i) for i in results])

**LeetCode 49. Group Anagrams**

class Solution:

def groupAnagrams(self, strs):

"""

:type strs: List[str]

:rtype: List[List[str]]

"""

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

return []

results = []

hash = dict()

for str in strs:

sig = ''.join(sorted([char for char in str]))

if sig not in hash:

hash[sig] = [str]

else:

hash[sig].append(str)

for sig in hash:

results.append(hash[sig])

return results

**LeetCode 224. Basic Calculator**

class Solution:

def calculator(self, s):

i = 0

n = len(s)

stack = []

result = 0

sign = 1

while i < n:

char = s[i]

if char == ‘+’:

sign = 1

i += 1

elif char == ‘-’:

sign = -1

i += 1

elif char == ‘(‘:

stack.append((sign, result))

result = 0

sign = 1

i += 1

elif char == ‘)’:

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

result \*= pre\_sign

result += pre\_result

i += 1

elif char == ‘ ‘:

i += 1

else:

temp = 0

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

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

i += 1

result += sign \* temp

return result

**LeetCode 791. Custom Sort String**

import collections

class Solution:

def custom\_sort\_string(self, S, T):

count = collections.Counter(T)

ans = []

for c in S:  
 ans.append(count[c] \* c)

count[c] = 0

for c in count:

ans.append(count[c] \* c)

return ‘’.join(ans)

**LeetCode 678. Valid Parenthesis String**

cmax counts the maximum open parenthesis,

which means the maximum number of unbalanced '(' that COULD be paired.

cmin counts the minimum open parenthesis,

which means the number of unbalanced '(' that MUST be paired.

class Solution:

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

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

return True

cmin = 0

cmax = 0

for c in s:

if c == ‘(‘:

cmin += 1

cmax += 1

if c == ‘)’:

cmin = max(cmin-1, 0)

cmax -= 1

if c == ‘\*’:

cmin = max(cmin-1, 0)

cmax += 1

if cmax < 0:

return False

return cmin == 0

**LeetCode 449. Serialize and Deserialize BST**

class Solution:

def serialize(self, root):

vals = []

def pre\_order(node):

val.append(node.val)

pre\_order(node.left)

pre\_order(node.right)

pre\_order(root)

return ‘,’.join([str(i] for i in vals])

def de\_serialize(self, data):

vals = iter(int(i) for i in data.split(‘,’))

def build(min\_val, max\_val):

if vals and min\_val < vals[0] and max\_val:

val = vals.pop(0)

node = TreeNode(val)

node.left = build(min\_val, val)

node.right = build(val, max\_val)

return node

return build(-float(‘inf’), float(‘inf’)

**LeetCode 273. Integer 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)

elif num < 1000:

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

**LeetCode 192. Word Frequency**

**LeetCode 8. String to Integer (atoi)**

def class Solution:

def my\_atoi(self, str):

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

return 0

str = str.strip()

digits = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

if str == '' or str[0] not in ['+', '-'] + digits:

return 0

flag = 1

result = 0

if str[0] == '+':

str = str[1:]

elif str[0] == '-':

flag = -1

str = str[1:]

for char in str:

if char not in digits:

break

result = result \* 10 + ord(char) - ord('0')

result \*= flag

if result < -2147483648:

return -2147483648

if result > 2147483647:

return 2147483647

return result

**LeetCode 10. Regular Expression Matching**

class Solution:

'.' Matches any single character.

'\*' Matches zero or more of the preceding element.

def is\_match(self, s, p):

m = len(s)

n = len(p)

dp = [[False for j in range(n+1)] for i in range(m+1)]

dp[0][0] = True

# \* = 0

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

dp[0][i] = dp[0][i-2] and p[i-1] == ‘\*’

for i in range(1, m+1):

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

if p[j-1] != ‘\*’:

dp[i][j] = dp[i-1][j-1] and (s[i-1] == p[j-1] or p[j-1] == ‘.’)

else:

dp[i][j] = dp[i][j-1] # \* = 1

if j >= 2:

dp[i][j] |= dp[i][j-2] # \* = 0

if s[i-1] == p[j-2] or p[j-2] == ‘.’:

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

return dp[-1][-1]

**LeetCode 301. Remove Invalid Parentheses**

class Solution:

def remove\_invalid\_parentheses(self, s):

results = []

visited = set()

self.remove(s, results, visited)

return results

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

mi = self.calc(s)

if mi == 0:

results.append(s)

return

n = len(s)

for i in range(n):

new\_s = s[:i] + s[i+1:]

if new\_s in visited:

continue

visited.add(new\_s)

new\_mi = self.calc(new\_s)

if new\_mi < mi:

self.remove(new\_s, results, visited)

def calc(self, s):

l = 0

r = 0

for c in s:

if c == '(':

l += 1

elif c == ')':

l -= 1

if l < 0:

r += 1

l = 0

return l + r

**LeetCode 51. N-Queens**

class Solution:

def solve\_N\_queens(self, n):

results = []

xy\_sum = set()

xy\_diff = set()

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

anses = []

for result in results:

ans = []

for q in result:

ans.append('.' \* q + 'Q' + '.' \* (n - 1 - q))

anses.append(ans)

return anses

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

**LeetCode 79. Word Search**

class Solution:

def exist(board, word):

m = len(board)

n = len(board[0])

for i in range(m):

for j in range(n):

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

return True

return False

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

if len(word) == 0:

return True

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

return False

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

return False

save = board[i][j]

board[i][j] = ‘.’

result = (

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

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

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

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

)

board[i][j] = save

return result

**LeetCode 785. Is Graph Bipartite?**

class Solution:

def isBipartite(self, graph: List[List[int]]) -> bool:

color = dict()

n = len(graph)

for node in range(n):

if node in color:

continue

stack = [node]

color[node] = 0

while stack:

node = stack.pop()

for nei in graph[node]:

if nei not in color:

stack.append(nei)

color[nei] = 1 - color[node]

elif color[nei] == color[node]:

return False

return True

**LeetCode 494. Target Sum**

class Solution:

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

memo = dict()

n = len(nums)

for i in range(n):

memo[i] = dict()

return self.calc(nums, 0, 0, S, memo)

def calc(self, nums, i, sum, S, memo):

if i == len(nums):

if sum == S:

return 1

else:

return 0

if sum in memo[i]:

return memo[i][sum]

add = self.calc(nums, i+1, sum+nums[i], S, memo)

sub = self.calc(nums, i+1, sum-nums[i], S, memo)

memo[i][sum] = add + sub

return memo[i][sum]

**LeetCode 129. Sum Root to Leaf Numbers**

class Solution:

def sumNumbers(self, root):

"""

:type root: TreeNode

:rtype: int

"""

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

path.append(root.val)

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

results.append(path[:])

self.dfs(root.left, path, results)

self.dfs(root.right, path, results)

path.pop()

**LeetCode 282. Expression Add Operators**

class Solution:

def add\_operator(self, num, target):

results = []

for i in range(1, len(num) + 1):

if i == 1 or (i > 1 and num[0] != ‘0’):

self.helper(num[i:], target, num[:i], int(num[:i]), int(num[:i]), results)

return results

def helper(self, num, target, path, cur, last, results):

if len(num) == 0:

if cur == target:

results.append(path)

return

for i in range(1, len(num) + 1):

val = num[:i]

if i == 1 or (i > 1 and num[0] != ‘0’):

self.helper(num[i:], target, path+‘+’+val, cur+int(val), int(val), results)

self.helper(num[i:], target, path+’-’+val, cur-int(val), -int(val), results)

self.helper(num[i:], target, path+’\*’+val, cur-last+last\*int(val), last\*int(val), results)

**LeetCode 37. Sudoku Solver**

class Solution:

def solve\_sudoku(self, board):

if board is None:

return

self.helper(board)

def helper(self, board):

for i in range(9):

for j in range(9):

if board[i][j] != ‘.’;

continue

for char in [str(x) for x in range(1,10)]:

if self.valid(board, i, j, char):

board[i][j] = char

if self.helper(board):

return True

else:

board[i][j] = ‘.’

return False

return True

def valid(self, board, row, col, char):

for i in range(9):

if board[i][col] == char:

return False

if board[row][i] == char:

return False

if board[3\*(row//3) + i//3][3\*(col//3) + i%3] == char:

return False

return True

**LeetCode 240. Search a 2D Matrix II**

class Solution:

def search\_matrix(self, matrix, target):

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

return False

m = len(matrix)

n = len(matrix[0])

i = 0

j = n-1

while i < m and j >= 0:

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

return True

elif matrix[i][j] > target:

j -= 1

elif matrix[i][j] < target:

i += 1

return False

**LeetCode 278. First Bad Version**

class Solution(object):

def firstBadVersion(self, n):

"""

:type n: int

:rtype: int

"""

l = 1

r = n

while l + 1 < r:

m = (l + r) // 2

if isBadVersion(m):

r = m

else:

l = m

if isBadVersion(l):

return l

if isBadVersion(r):

return r

return 0

**LeetCode 29. Divide Two Integers**

class Solution:

def divide(self, dividend, divisor):

if divisor == 0:

return 2147483647

flag = 1

if (dividend > 0 and divisor < 0) or (divided < 0 and divisor > 0):

flag = -1

dividend = abs(dividend)

divisor = abs(dividend)

result = 0

while dividend >= divisor:

i = 0

while dividend >= (divisor << i):

i += 1

result += 1 << (i - 1)

dividend -= divisor << (i-1)

result \*= flag

if result < -2147483648 or result > 2147483647:

return 2147483647

return result

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

class Solution:

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

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:

r = m - 1

else:

l = m

if nums[r] == target:

results.append(r)

elif nums[l] == target:

results.append(l)

else:

return [-1, -1]

return results

**LeetCode 21. Merge Two Sorted Lists**

class Solution(object):

def mergeTwoLists(self, l1, l2):

dummy = ListNode(-1)

cur = dummy

l1\_cur = l1

l2\_cur = l2

while l1\_cur is not None and l2\_cur is not None:

if l1\_cur.val <= l2\_cur.val:

cur.next = l1\_cur

l1\_cur = l1\_cur.next

else:

cur.next = l2\_cur

l2\_cur = l2\_cur.next

cur = cur.next

if l1\_cur is not None:

cur.next = l1\_cur

if l2\_cur is not None:

cur.next = l2\_cur

return dummy.next

**LeetCode 92. Reverse Linked List II**

class Solution:

def reverse\_between(self, head, m, n):

if head is None:

return head

dummy = ListNode(-1)

dummy.next = head

lead = dummy

tail = dummy

for i in range(m-1):

lead = lead.next

for i in range(n):

tail = tail.next

new\_lead = lead.next

new\_tail = tail.next

tail.next = None

before, after = self.helper(new\_lead)

lead.next = before

after.next = new\_tail

return dummy.next

def helper(self, head):

after = head

before = None

current = head

while current:

next = current.next

current.next = before

before = current

current = next

return before, after

**LeetCode 143. Reorder List**

class Solution:

def reorder\_list(self, head):

if head is None or head.next is None:

return head

nodes = []

now = head

while now:

node.append(now)

now = now.next

dummy = ListNode(-1)

now = dummy

i = 0

j = len(nodes) - 1

while i < j:

now.next = nodes[i]

now.next.next = nodes[j]

now = now.next.next

i += 1

j -= 1

if i == j:

now.next = nodes[i]

now.next.next = None

else:

now.next = None

**LeetCode 23. Merge K Sorted Lists**

import heapq

class Solution:

def merge\_k\_lists(self, lists):

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

return None

heap = []

heap.heapify(heap)

for node in lists:

while node is not None:

heaqp.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 109. Convert Sorted List to Binary Search Tree**

class Solution:

def sorted\_list\_to\_bst(self, head):

if head is None:

return None

if head.next is None:

return TreeNode(head.val)

dummy = ListNode(-1)

dummy.next = head

slow = dummy

fast = dummy

while fast and fast.next:

fast = fast.next.next

if fast is not None:

slow = slow.next

center = slow.next

slow.next = none

node = TreeNode(center.val)

node.left = self.sorted\_list\_to\_bst(head)

node.right = self.sorted\_list\_to\_bst(center.next)

return node

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

class Solution:

def tree\_to\_doubly\_list(self, root):

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)

**LeetCode 146. LRU Cache**

class Node:

def \_\_init\_\_(self, key, val):

self.key = key

self.val = val

self.prev = None

self.next = None

class LRU:

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

self.capacity = capacity

self.dict = dict()

self.head = Node(0, 0)

self.tail = Node(0, 0)

self.head.next = self.tail

self.tail.prev = self.head

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, val):

if key in self.dict:

n = self.dict[key]

self.\_remove(n)

n = Node(val)

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.prev = p

self.tail.prev = n

n.next = self.tail

def \_remove(self, n):

prev = n.prev

nex = n.next

prev.next = nex

nex.prev = prev

**LeetCode 97. Interleaving String**

class Solution:

def is\_interleave(self, s1, s2, s3):

m = len(s1)

n = len(s2)

if m == 0:

return s2 == s3

if n == 0:

return s1 == s3

if m + n != len(s3):

return False

dp = [[False] \* (n + 1) for \_ in range(m+1)]

for i in range(m+1):

for j in range(n+1):

if i == 0 and j == 0:

dp[i][j] = True

elif i == 0:

dp[i][j] = (s2[:j] == s3[:j])

elif j == 0:

dp[i][j] = (s1[:i] == s3[:i])

else:

dp[i][j] = (dp[i-1][j] and s1[i-1] == s3[i+j-1]) or (d[i][j-1] and s2[j-1] == s3[i+j-1])

return dp[-1][-1]

**300. Longest Increasing Subsequence**

class Solution:

def length\_of\_LIS(self, nums):

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

return 0

n = len(nums)

dp = [1] \* n

max\_lis = 1

for i in range(1, n):

for j in range(i):

if nums[i] > nums[j]:

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

max\_lis = max(max\_lis, dp[i])

return max\_lis

**LeetCode 416. Partition Equal Subset Sum**

class Solution:

def can\_partition(self, nums):

num\_sum = sum(nums)

if num\_sum % 2 == 1:

return False

half = num\_sum // 2

access = [False] \* (half + 1)

access[0] = True

for num in nums:

for i in reversed(range(num, half+1)):

if access[i-num]:

access[i] = True

return access[-1]

**LeetCode 639. Decode Ways II**

class Solution:

def num\_decodings(self, s):

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

e0, e1, e2 = f0, f1, f2

return e0

**LeetCode 664. Strange Printer**

class Solution:

def strange\_printer(self, 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):

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 903. Valid Permutations for DI Sequence**

class Solution:

def accont\_merge(self, accounts):

self.root = dict()

em2id = dict()

em2name = dict()

i = 0

for acc in accounts:

name = acc[0]

for em in acc[1:]:

em2id[em] = i

em2name[em] = name

self.root[i] = i

i += 1

ans = collections.defaultdict(list)

for em in em2id:

ans[self.find(em2id[em])].append(em)

return [[em2name[v[0]]] + sorted(v) for v in ans.values()]

def find(self, x):

if self.root[x] != x:

self.root[x] = self.find(self.root[x])

return self.root[x]

def union(self, x, y):

self.root[self.find(x)] = self.find(y)

**LeetCode 399. Evaluate Division**

class Solution:

def calc\_equation(self, equations, values, queries):

n = len(equations)

self.root = dict()

for i in range(n):

a = equations[i][0]

b = equations[i][1]

self.root[a] = [a, 1]

self.root[b] = [b, 1]

for i in range(n):

a = equations[i][0]

b = equations[i][1]

self.union(a, b, values[i])

results = []

for query in queries:

a = query[0]

b = query[1]

a\_root, a\_times = self.find(a)

b\_root, b\_times = self.find(b)

if a\_root != b\_root or a\_root == '\*' or b\_root == '\*':

results.append(-1.0)

else:

results.append(a\_times/b\_times)

return results

def union(self, a, b, value):

a\_root, a\_times = self.find(a)

b\_root, b\_times = self.find(b)

self.root[a\_root] = [b\_root, value \* b\_times / a\_times]

def find(self, a):

if a not in self.root:

return ['\*', 1]

a\_root, a\_times = self.root[a]

if a\_root != a:

a\_root\_root, a\_root\_times = self.find(a\_root)

self.root[a] = [a\_root\_root, a\_times \* a\_root\_times]

return self.root[a]

**LeetCode 263. Ugly Number**

class Solution:

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

if num <= 0:

return False

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

return True

if num % 2 == 0:

return self.isUgly(num//2)

elif num % 3 == 0:

return self.isUgly(num//3)

elif num % 5 == 0:

return self.isUgly(num//5)

else:

return False

**LeetCode 794. Valid Tic-Tac-Toe State**

class Solution:

def validTicTacToe(self, board: List[str]) -> bool:

first = 'X'

second = 'O'

x\_count = sum(row.count('X') for row in board)

o\_count = sum(row.count('O') for row in board)

print(x\_count, o\_count)

def win(board, player):

for i in range(3):

if all(board[i][j] == player for j in range(3)):

return True

if all(board[j][i] == player for j in range(3)):

return True

if player == board[0][0] == board[1][1] == board[2][2]:

return True

if player == board[0][2] == board[1][1] == board[2][0]:

return True

return False

if o\_count not in set([x\_count-1, x\_count]):

return False

if win(board, first) and x\_count-1 != o\_count:

return False

if win(board, second) and x\_count != o\_count:

return False

return True

**LeetCode 670. Maximum Swap**

class Solution:

def maximumSwap(self, num):

"""

:type num: int

:rtype: int

"""

num\_str = str(num)

num\_list = [char for char in num\_str]

num\_list\_sorted = sorted(num\_list, reverse=True)

index = 0

for i in range(len(num\_str)):

if num\_list[i] != num\_list\_sorted[i]:

index = i

break

max\_index = index

for i in range(index+1, len(num\_str)):

if num\_list[i] >= num\_list[max\_index]:

max\_index = i

num\_list[index], num\_list[max\_index] = num\_list[max\_index], num\_list[index]

# print(num\_str, num\_list, num\_list\_sorted)

return int(''.join(num\_list))

**LeetCode 20. Valid Parentheses**

class Solution:

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

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

return True

n = len(s)

if n % 2 == 1:

return False

stack = []

for c in s:

if c in set(['(', '[', '{']):

stack.append(c)

else:

if len(stack) > 0:

top = stack.pop()

if (top == '(' and c == ')') or (top == '{' and c == '}') or (top == '[' and c == ']'):

continue

else:

return False

else:

return False

return len(stack) == 0

**LeetCode 844. Backspace String Compare**

class Solution:

def backspaceCompare(self, S: str, T: str) -> bool:

s\_l = self.to\_list(S)

t\_l = self.to\_list(T)

if s\_l == t\_l:

return True

return False

def to\_list(self, S):

s\_l = []

for c in S:

if c != '#':

s\_l.append(c)

else:

if len(s\_l) > 0:

s\_l.pop()

return s\_l

**LeetCode 480. Sliding Window Median**

def medianSlidingWindow(nums, k):

small, large = [], []

for i, n in enumerate(nums[:k]):

heapq.heappush(small, (-n,i))

for \_ in range(k-k//2):

move(small, large)

ans = [get\_med(small, large, k)]

for i, n in enumerate(nums[k:]):

if n >= large[0][0]:

heapq.heappush(large, (n, i+k))

if nums[i] <= large[0][0]: move(large, small)

else:

heapq.heappush(small, (-n, i+k))

if nums[i] >= large[0][0]:

move(small, large)

while small and small[0][1] <= i:

heapq.heappop(small)

while large and large[0][1] <= i:

heapq.heappop(large)

ans.append(get\_med(small, large, k))

return ans

def move(h1, h2):

x, i = heapq.heappop(h1)

heapq.heappush(h2, (-x, i))

def get\_med(h1, h2, k):

return h2[0][0] \* 1. if k & 1 else (h2[0][0]-h1[0][0]) / 2.

**LeetCode 211. Add and Search Word - Data structure design**

class WordDictionary:

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

"""

Initialize your data structure here.

"""

self.root = Node()

def addWord(self, word):

"""

Adds a word into the data structure.

:type word: str

:rtype: void

"""

cur = self.root

for char in word:

if char not in cur.children:

cur.children[char] = Node()

cur = cur.children[char]

cur.is\_word = True

def search(self, word):

"""

Returns if the word is in the data structure. A word could contain the dot character '.' to represent any one letter.

:type word: str

:rtype: bool

"""

return self.helper(word, self.root)

def helper(self, word, node):

for char in word:

if char == '.':

for child in node.children:

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

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

**LeetCode 348. Design Tic-Tac-Toe**

Record the number of moves for each rows, columns, and two diagonals.

For each move, we -1 for each player 1's move and +1 for player 2's move.

Then we just need to check whether any of the recored numbers equal to n or -n.

class TicTacToe(object):

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

self.row, self.col, self.diag, self.anti\_diag, self.n = [0] \* n, [0] \* n, 0, 0, n

def move(self, row, col, player):

offset = player \* 2 - 3

self.row[row] += offset

self.col[col] += offset

if row == col:

self.diag += offset

if row + col == self.n - 1:

self.anti\_diag += offset

if self.n in [self.row[row], self.col[col], self.diag, self.anti\_diag]:

return 2

if -self.n in [self.row[row], self.col[col], self.diag, self.anti\_diag]:

return 1

return 0

**LeetCode 269. Alien Dictionary**

class Solution:

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

IN = dict()

OUT = dict()

for word in words:

for ch in word:

if ch not in IN:

IN[ch] = set()

if ch not in OUT:

OUT[ch] = set()

n = len(words)

for i in range(n-1):

word1 = words[i]

word2 = words[i+1]

k = 0

while k < len(word1) and k < len(word2):

if word1[k] != word2[k]:

IN[word2[k]].add(word1[k])

OUT[word1[k]].add(word2[k])

break

k += 1

print(IN)

print(OUT)

queue = []

results = []

for ch in IN:

if len(IN[ch]) == 0:

queue.append(ch)

print(queue)

if len(queue) == 0:

return ''

while len(queue) > 0:

ch = queue.pop(0)

results.append(ch)

for n in OUT[ch]:

IN[n].remove(ch)

if len(IN[n]) == 0:

queue.append(n)

for ch in IN:

if len(IN[ch]) > 0:

return ''

return ''.join(results)

**[HARD] LeetCode 358. Rearrange String k Distance Apart**

The idea is simple: we only worry about the most frequent character(s).

For example aaaabbbbcccddefg, ais the most frequent letter, so we start with a structure like

a [] a [] a [] a []

from collections import defaultdict

class Solution(object):

def rearrangeString(self, string, k):

if not string:

return ''

count = defaultdict(int)

for s in string:

count[s] += 1

# sort the letters according to the frequency

stack = sorted(list(count.items()), key=lambda t: t[1])

char, count = stack.pop() # get most frequent character

lst = [[char] for \_ in range(count)]

# take care of the letters with same highest freq

while stack and stack[-1][1] == count:

char, \_ = stack.pop()

for l in lst:

l.append(char)

# all the characters left

res = ''.join(c\*n for c, n in stack)

# padding

for i, r in enumerate(res):

lst[i % (len(lst)-1)].append(r)

for l in lst[:-1]:

if len(l) < k:

return ''

return ''.join(''.join(l) for l in lst)

**[HARD] LeetCode 126. Word Ladder II**

class Solution(object):

def findLadders(self, beginWord, endWord, wordList):

wordList = set(wordList)

res = []

layer = {}

layer[beginWord] = [[beginWord]]

while layer:

newlayer = collections.defaultdict(list)

for w in layer:

if w == endWord:

res.extend(k for k in layer[w])

else:

for i in range(len(w)):

for c in 'abcdefghijklmnopqrstuvwxyz':

neww = w[:i]+c+w[i+1:]

if neww in wordList:

newlayer[neww]+=[j+[neww] for j in layer[w]]

wordList -= set(newlayer.keys())

layer = newlayer

return res

**LeetCode 378. Kth Smallest Element in a Sorted Matrix**

class Solution(object):

def kthSmallest(self, matrix, k):

result, heap = None, []

heapq.heappush(heap, (matrix[0][0], 0, 0))

while k > 0:

result, i, j = heapq.heappop(heap)

if i == 0 and j + 1 < len(matrix):

heapq.heappush(heap, (matrix[i][j + 1], i, j + 1))

if i + 1 < len(matrix):

heapq.heappush(heap, (matrix[i + 1][j], i + 1, j))

k -= 1

return result

**LeetCode 903. Valid Permutations for DI Sequence**

Intuition:

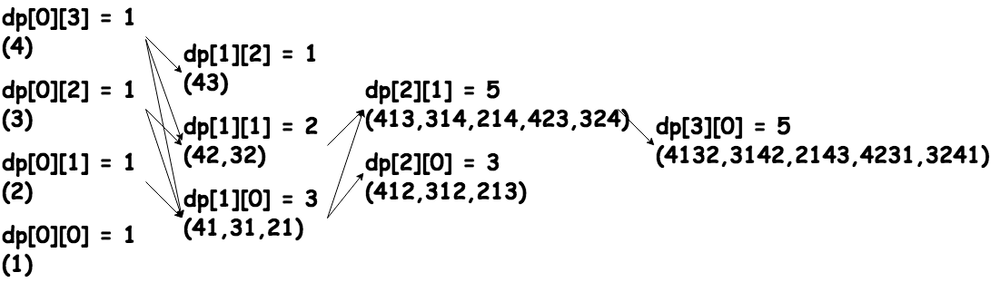
This the only problem this week that I feel like writing a solution.

But don't know how to explain.

dp[i][j] means the number of possible permutations of first i + 1 digits,

where the i + 1th digit is j + 1th smallest in the rest of digits.

Ok, may not make sense ... Let's see the following diagram.



I take the example of S = "DID".

The permutation can start from 1, 2, 3, 4.

So dp[0][0] = dp[0][1] = dp[0][2] = dp[0][3] = 1.

In the parenthesis, I list all possible permutations.

We decrese from the first digit to the second,

the down arrow show the all possibile decresing pathes.

The same, cause we increase from the second digit to the third,

the up arrow show the all possibile increasing pathes.

dp[2][1] = 5, mean the number of permutations

where the third digitis the second smallest of the rest.

We have 413,314,214,423,324.

Fow example 413, where 2,3 are left and 3 the second smallest of them.

Explanation:

As shown in the diagram,

for "I", we calculate prefix sum of the array,

for "D", we calculate sufixsum of the array.

Time Complexity:

O(N^2)

C++:

int numPermsDISequence(string S) {

int n = S.length(), mod = 1e9 + 7;

vector<vector<int>> dp(n + 1, vector<int>(n + 1));

for (int j = 0; j <= n; j++) dp[0][j] = 1;

for (int i = 0; i < n; i++)

if (S[i] == 'I')

for (int j = 0, cur = 0; j < n - i; j++)

dp[i + 1][j] = cur = (cur + dp[i][j]) % mod;

else

for (int j = n - i - 1, cur = 0; j >= 0; j--)

dp[i + 1][j] = cur = (cur + dp[i][j + 1]) % mod;

return dp[n][0];

}

Java:

public int numPermsDISequence(String S) {

int n = S.length(), mod = (int)1e9 + 7;

int[][] dp = new int[n + 1][n + 1];

for (int j = 0; j <= n; j++) dp[0][j] = 1;

for (int i = 0; i < n; i++)

if (S.charAt(i) == 'I')

for (int j = 0, cur = 0; j < n - i; j++)

dp[i + 1][j] = cur = (cur + dp[i][j]) % mod;

else

for (int j = n - i - 1, cur = 0; j >= 0; j--)

dp[i + 1][j] = cur = (cur + dp[i][j + 1]) % mod;

return dp[n][0];

}

Now as we did for every DP, make it 1D dp.

Reminded by @apple702, in the Java solution, it should be dp=Arrays.copyOf(dp2, n);

Otherwise it passes an address.

C++:

int numPermsDISequence(string S) {

int n = S.length(), mod = 1e9 + 7;

vector<int> dp(n + 1, 1), dp2(n);

for (int i = 0; i < n; dp = dp2, i++) {

if (S[i] == 'I')

for (int j = 0, cur = 0; j < n - i; j++)

dp2[j] = cur = (cur + dp[j]) % mod;

else

for (int j = n - i - 1, cur = 0; j >= 0; j--)

dp2[j] = cur = (cur + dp[j + 1]) % mod;

}

return dp[0];

}

Java:

public int numPermsDISequence(String S) {

int n = S.length(), mod = (int)1e9 + 7;

int[] dp = new int[n + 1], dp2 = new int[n];;

for (int j = 0; j <= n; j++) dp[j] = 1;

for (int i = 0; i < n; i++) {

if (S.charAt(i) == 'I')

for (int j = 0, cur = 0; j < n - i; j++)

dp2[j] = cur = (cur + dp[j]) % mod;

else

for (int j = n - i - 1, cur = 0; j >= 0; j--)

dp2[j] = cur = (cur + dp[j + 1]) % mod;

dp = Arrays.copyOf(dp2, n);

}

return dp[0];

}

Python:

def numPermsDISequence(self, S):

dp = [1] \* (len(S) + 1)

for c in S:

if c == "I":

dp = dp[:-1]

for i in range(1, len(dp)):

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

else:

dp = dp[1:]

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

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

return dp[0] % (10\*\*9 + 7)