Count=329

Optimaizations needed:

2248,2219,2210,2200,2194,2181,2180,2176,2165,2161,2130,2128,2125,2104,2103,

2091,2085,2083,2078,2068,2062,1852,2046,2042,2023,2022,2016,2006,1358,2248,1995,1901,1880,1812 , 1837,1698,1602,1512,1539,1566,1570,1474,1490,1346,1299,1213,1180,1143,1071,1079, 1089,1099

1001-1050 ($$$$$ not done)

**1002. Find Common Characters**

Given a string array words, return *an array of all characters that show up in all strings within the*words*(including duplicates)*. You may return the answer in **any order**.

**Example 1:**

**Input:** words = ["bella","label","roller"]

**Output:** ["e","l","l"]

**My sol:**

class Solution:

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

p=0

temp=words[0]

# print(type(temp))

for x in words[1:]:

for y in temp :

if y in x and x.count(y)>=temp.count(y):

pass

else:

print("test")

print(y)

temp=temp.replace(y,"",1)

return temp

**1004. Max Consecutive Ones III**

Given a binary array nums and an integer k, return *the maximum number of consecutive*1*'s in the array if you can flip at most* k 0's.

**Example 1:**

**Input:** nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2

**Output:** 6

**Explanation:** [1,1,1,0,0,**1**,1,1,1,1,**1**]

Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

**My sol:**

class Solution:

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

i=0

result=0

for j in range(0,len(nums)):

if nums[j]==0:

k-=1

while k<0:

if nums[i]==0:

k+=1

i+=1

result=max(result,j-i+1)

return result

**1009. Complement of Base 10 Integer**

The **complement** of an integer is the integer you get when you flip all the 0's to 1's and all the 1's to 0's in its binary representation.

* For example, The integer 5 is "101" in binary and its **complement** is "010" which is the integer 2.

Given an integer n, return *its complement*.

**Example 1:**

**Input:** n = 5

**Output:** 2

**Explanation:** 5 is "101" in binary, with complement "010" in binary, which is 2 in base-10.

**My sol:**

class Solution:

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

s=""

a=bin(n)

print(a)

b=a[2:]

for x in b:

if x=="1":

s=s+"0"

else:

s=s+"1"

return int(s,2)

**1016. Binary String With Substrings Representing 1 To N**

Given a binary string s and a positive integer n, return true*if the binary representation of all the integers in the range*[1, n]*are****substrings****of*s*, or*false*otherwise*.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** s = "0110", n = 3

**Output:** true

**My sol:**

class Solution:

def queryString(self, s: str, n: int) -> bool:

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

if bin(i)[2:] not in s:

return False

return True

**1017. Convert to Base -2**

Given an integer n, return *a binary string representing its representation in base* -2.

**Note** that the returned string should not have leading zeros unless the string is "0".

**Example 1:**

**Input:** n = 2

**Output:** "110"

**Explantion:** (-2)2 + (-2)1 = 2

**My sol:**

import numpy as np

class Solution:

def baseNeg2(self, n: int) -> str:

if n in [0, 1]: return str(n)

if n % 2 == 0:

return self.baseNeg2(n // -2) + '0'

else:

return self.baseNeg2((n - 1) // -2) + '1'

**1018. Binary Prefix Divisible By 5**

You are given a binary array nums (**0-indexed**).

We define xi as the number whose binary representation is the subarray nums[0..i] (from most-significant-bit to least-significant-bit).

* For example, if nums = [1,0,1], then x0 = 1, x1 = 2, and x2 = 5.

Return *an array of booleans*answer*where*answer[i]*is*true*if*xi*is divisible by*5.

**Example 1:**

**Input:** nums = [0,1,1]

**Output:** [true,false,false]

**Explanation:** The input numbers in binary are 0, 01, 011; which are 0, 1, and 3 in base-10.

Only the first number is divisible by 5, so answer[0] is true.

**My sol:**  
import numpy

class Solution:

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

a=[]

s=""

for x in range(0,len(nums)):

s+=str(nums[x])

a.append(s)

temp=[]

for x in a:

temp.append(int(x,2)%5==0)

return temp

**1021. Remove Outermost Parentheses**

A valid parentheses string is either empty "", "(" + A + ")", or A + B, where A and B are valid parentheses strings, and + represents string concatenation.

* For example, "", "()", "(())()", and "(()(()))" are all valid parentheses strings.

A valid parentheses string s is primitive if it is nonempty, and there does not exist a way to split it into s = A + B, with A and B nonempty valid parentheses strings.

Given a valid parentheses string s, consider its primitive decomposition: s = P1 + P2 + ... + Pk, where Pi are primitive valid parentheses strings.

Return s *after removing the outermost parentheses of every primitive string in the primitive decomposition of*s.

**Example 1:**

**Input:** s = "(()())(())"

**Output:** "()()()"

**Explanation:**

The input string is "(()())(())", with primitive decomposition "(()())" + "(())".

After removing outer parentheses of each part, this is "()()" + "()" = "()()()".

**My sol:**

class Solution:

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

ans = current = ""

stack = []

for ch in S:

if stack and stack[-1] == '(' and ch == ')':

stack.pop()

else:

stack.append(ch)

current += ch

if not stack:

ans += current[1:-1]

current = ""

return ans

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

You are given the root of a binary tree where each node has a value 0 or 1. Each root-to-leaf path represents a binary number starting with the most significant bit.

* For example, if the path is 0 -> 1 -> 1 -> 0 -> 1, then this could represent 01101 in binary, which is 13.

For all leaves in the tree, consider the numbers represented by the path from the root to that leaf. Return *the sum of these numbers*.

The test cases are generated so that the answer fits in a **32-bits** integer.

**Example 1:**

Shape, arrow

Description automatically generated

**Input:** root = [1,0,1,0,1,0,1]

**Output:** 22

**Explanation:** (100) + (101) + (110) + (111) = 4 + 5 + 6 + 7 = 22

**My sol:**

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

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

def temp(root,path):

if root:

path+=str(root.val)

if not root.left and not root.right:

paths.append(path)

else:

path+=''

temp(root.left,path)

temp(root.right,path)

paths=[]

temp(root,'')

count=0

# print(bin(4))

for x in paths:

# print(int(x,2))

count+=int(x,2)

return count

**1025. Divisor Game**

Alice and Bob take turns playing a game, with Alice starting first.

Initially, there is a number n on the chalkboard. On each player's turn, that player makes a move consisting of:

* Choosing any x with 0 < x < n and n % x == 0.
* Replacing the number n on the chalkboard with n - x.

Also, if a player cannot make a move, they lose the game.

Return true *if and only if Alice wins the game, assuming both players play optimally*.

**Example 1:**

**Input:** n = 2

**Output:** true

**Explanation:** Alice chooses 1, and Bob has no more moves.

**My sol:**

class Solution:

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

# a=[]

# while n>0:

# a.append(1)

# n-=1

# if len(a)%2==0:

# return True

# else:

# return False

P = [False for i in range(n + 1)]

print(P)

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

for j in range(1, i):

if i % j == 0 and P[i - j] == False:

P[i] = True

print(P)

return P[n]

**1030. Matrix Cells in Distance Order**

You are given four integers row, cols, rCenter, and cCenter. There is a rows x cols matrix and you are on the cell with the coordinates (rCenter, cCenter).

Return *the coordinates of all cells in the matrix, sorted by their****distance****from*(rCenter, cCenter)*from the smallest distance to the largest distance*. You may return the answer in **any order** that satisfies this condition.

The **distance** between two cells (r1, c1) and (r2, c2) is |r1 - r2| + |c1 - c2|.

**Example 1:**

**Input:** rows = 1, cols = 2, rCenter = 0, cCenter = 0

**Output:** [[0,0],[0,1]]

**Explanation:** The distances from (0, 0) to other cells are: [0,1]

**My sol:**

class Solution:

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

temp=[]

for x in range(0,rows):

for y in range(0,cols):

temp.append([x,y])

a=sorted(temp, key=lambda x: ((abs(rCenter-x[0])+abs(cCenter-x[1])), x))

return a

**1037. Valid Boomerang**

Given an array points where points[i] = [xi, yi] represents a point on the **X-Y** plane, return true *if these points are a****boomerang***.

A **boomerang** is a set of three points that are **all distinct** and **not in a straight line**.

**Example 1:**

**Input:** points = [[1,1],[2,3],[3,2]]

**Output:** true

**My sol:**

class Solution:

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

flag=0

x0, y0 = points[0]

x1, y1 = points[1]

dy = y1 - y0

dx = x1 - x0

xp = dx\*y0 - dy\*x0

print(xp)

for x, y in points :

if dx\*y - dy\*x == xp:

flag=0

else:

flag=1

break

return flag==1

**1046. Last Stone Weight**

You are given an array of integers stones where stones[i] is the weight of the ith stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with x <= y. The result of this smash is:

* If x == y, both stones are destroyed, and
* If x != y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return 0.

**Example 1:**

**Input:** stones = [2,7,4,1,8,1]

**Output:** 1

**Explanation:**

We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,

we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,

we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,

we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of the last stone.

**My sol:**

class Solution:

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

if len(stones)==2 and len(set(stones))==1:

return 0

while len(stones)>1:

if len(stones)==2 and len(set(stones))==1:

return 0

a=sorted(stones)[::-1]

temp1=a[0]

temp2=a[1]

# print(temp1)

# print(temp2)

if temp1==temp2:

stones.remove(temp1)

stones.remove(temp2)

else:

stones.remove(temp2)

for x in range(0,len(stones)):

if stones[x]==temp1:

stones[x]=abs(temp1-temp2)

print(stones)

return stones[0]

**1047. Remove All Adjacent Duplicates In String**

You are given a string s consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on s until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

**Example 1:**

**Input:** s = "abbaca"

**Output:** "ca"

**Explanation:**

For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is "aaca", of which only "aa" is possible, so the final string is "ca".

**My sol:**

class Solution:

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

stack=[]

a=''

for x in s:

if stack and stack[-1]==x:

stack.pop()

a=a[:-1]

else:

stack.append(x)

a+=x

return a

1051-1100 ($$$$$ 13)

**1051. Height Checker**

A school is trying to take an annual photo of all the students. The students are asked to stand in a single file line in **non-decreasing order** by height. Let this ordering be represented by the integer array expected where expected[i] is the expected height of the ith student in line.

You are given an integer array heights representing the **current order** that the students are standing in. Each heights[i] is the height of the ith student in line (**0-indexed**).

Return *the****number of indices****where*heights[i] != expected[i].

**Example 1:**

**Input:** heights = [1,1,4,2,1,3]

**Output:** 3

**Explanation:**

heights: [1,1,4,2,1,3]

expected: [1,1,1,2,3,4]

Indices 2, 4, and 5 do not match.

**My sol:**

class Solution:

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

count=0

b=sorted(heights)

for i in range(0,len(heights)):

if heights[i] != b[i]:

count+=1

return count

**1056. Confusing Number**

A **confusing number** is a number that when rotated 180 degrees becomes a different number with **each digit valid**.

We can rotate digits of a number by 180 degrees to form new digits.

* When 0, 1, 6, 8, and 9 are rotated 180 degrees, they become 0, 1, 9, 8, and 6 respectively.
* When 2, 3, 4, 5, and 7 are rotated 180 degrees, they become **invalid**.

Note that after rotating a number, we can ignore leading zeros.

* For example, after rotating 8000, we have 0008 which is considered as just 8.

Given an integer n, return true*if it is a****confusing number****, or*false*otherwise*.

**Example 1:**

A picture containing text, weapon, scissors, clipart

Description automatically generated

**Input:** n = 6

**Output:** true

**Explanation:** We get 9 after rotating 6, 9 is a valid number, and 9 != 6.

**My sol:**

class Solution:

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

dict={}

s=""

dict[0]='0'

dict[1]='1'

dict[2]='a'

dict[3]='b'

dict[4]='c'

dict[5]='d'

dict[6]='9'

dict[7]='e'

dict[8]='8'

dict[9]='6'

print(dict)

for x in str(n):

s+=str(dict[int(x)])

for x in s:

if x not in ['0','1','2','3','4','5','6','7','8','9']:

return False

return s[::-1]!=str(n)

**1064. Fixed Point**

Given an array of distinct integers arr, where arr is sorted in **ascending order**, return the smallest index i that satisfies arr[i] == i. If there is no such index, return -1.

**Example 1:**

**Input:** arr = [-10,-5,0,3,7]

**Output:** 3

**Explanation:** For the given array, arr[0] = -10, arr[1] = -5, arr[2] = 0, arr[3] = 3, thus the output is 3.

**My sol:**

class Solution:

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

a=[]

for i in range(0,len(arr)):

if arr[i]==i:

a.append(arr[i])

if a==[]:

return -1

else:

return min(a)

**1065. Index Pairs of a String**

Given a string text and an array of strings words, return *an array of all index pairs*[i, j]*so that the substring*text[i...j]*is in words*.

Return the pairs [i, j] in sorted order (i.e., sort them by their first coordinate, and in case of ties sort them by their second coordinate).

**Example 1:**

**Input:** text = "thestoryofleetcodeandme", words = ["story","fleet","leetcode"]

**Output:** [[3,7],[9,13],[10,17]]

My sol:

import re

class Solution:

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

temp=[]

for x in range(0,len(text)):

for y in words:

if text[x:x+len(y)]==y:

# print(x,x+len(y)-1)

temp.append([x,x+len(y)-1])

a = sorted(temp, key=lambda x: (x[0],x[1]))

return a

**1071. Greatest Common Divisor of Strings**

For two strings s and t, we say "t divides s" if and only if s = t + ... + t (i.e., t is concatenated with itself one or more times).

Given two strings str1 and str2, return *the largest string*x*such that*x*divides both*str1*and*str2.

**Example 1:**

**Input:** str1 = "ABCABC", str2 = "ABC"

**Output:** "ABC"

**My sol:**

class Solution:

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

return str1[:math.gcd(len(str1), len(str2))] if str1 + str2 == str2 + str1 else ''

**1079. Letter Tile Possibilities**

You have n  tiles, where each tile has one letter tiles[i] printed on it.

Return *the number of possible non-empty sequences of letters* you can make using the letters printed on those tiles.

**Example 1:**

**Input:** tiles = "AAB"

**Output:** 8

**Explanation:** The possible sequences are "A", "B", "AA", "AB", "BA", "AAB", "ABA", "BAA".

**My sol:**

class Solution:

def numTilePossibilities(self, tiles: str) -> int:

return sum(len(set(itertools.permutations(tiles, i))) for i in range(1, len(tiles) + 1))

**1081. Smallest Subsequence of Distinct Characters**

Given a string s, return *the lexicographically smallest subsequence of* s *that contains all the distinct characters of* s *exactly once*.

**Example 1:**

**Input:** s = "bcabc"

**Output:** "abc"

**My sol:**

class Solution:

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

res = []

for i in range(len(s)):

if s[i] in res:

continue

while res and s[i]<res[-1] and res[-1] in s[i+1:]:

res.pop()

res.append(s[i])

return "".join(res)

**1085. Sum of Digits in the Minimum Number**

Given an integer array nums, return 0*if the sum of the digits of the minimum integer in*nums*is odd, or*1*otherwise*.

**Example 1:**

**Input:** nums = [34,23,1,24,75,33,54,8]

**Output:** 0

**Explanation:** The minimal element is 1, and the sum of those digits is 1 which is odd, so the answer is 0.

**My sol:**

class Solution:

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

count=0

x=min(nums)

a=str(x)

for y in a:

count+=int(y)

if count%2!=0:

return 0

else:

return 1

**1086. High Five**

Given a list of the scores of different students, items, where items[i] = [IDi, scorei] represents one score from a student with IDi, calculate each student's **top five average**.

Return *the answer as an array of pairs*result*, where*result[j] = [IDj, topFiveAveragej]*represents the student with*IDj*and their****top five average****. Sort*result*by*IDj*in****increasing order****.*

A student's **top five average** is calculated by taking the sum of their top five scores and dividing it by 5 using **integer division**.

**Example 1:**

**Input:** items = [[1,91],[1,92],[2,93],[2,97],[1,60],[2,77],[1,65],[1,87],[1,100],[2,100],[2,76]]

**Output:** [[1,87],[2,88]]

**Explanation:**

The student with ID = 1 got scores 91, 92, 60, 65, 87, and 100. Their top five average is (100 + 92 + 91 + 87 + 65) / 5 = 87.

The student with ID = 2 got scores 93, 97, 77, 100, and 76. Their top five average is (100 + 97 + 93 + 77 + 76) / 5 = 88.6, but with integer division their average converts to 88.

My sol:

class Solution:

def highFive(self, items: List[List[int]]) -> List[List[int]]:

a=[]

b=[]

c=[]

dict={}

for x in items:

if x[0] not in a:

a.append(x[0])

print(a)

for x in a:

for y in items:

if y[0]==x:

b.append(y[1])

c.append([x,int(sum(sorted(b)[::-1][:5])/5)])

b=[]

print(c)

return sorted(c)

**1089. Duplicate Zeros**

Given a fixed-length integer array arr, duplicate each occurrence of zero, shifting the remaining elements to the right.

**Note** that elements beyond the length of the original array are not written. Do the above modifications to the input array in place and do not return anything.

**Example 1:**

**Input:** arr = [1,0,2,3,0,4,5,0]

**Output:** [1,0,0,2,3,0,0,4]

**Explanation:** After calling your function, the input array is modified to: [1,0,0,2,3,0,0,4]

**My sol:**

class Solution:

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

"""

Do not return anything, modify arr in-place instead.

"""

l=len(arr)

temp=[]

for x in arr:

if x ==0:

temp.append(x)

temp.append(x)

arr[:]=temp[:l]

**1094. Car Pooling**

There is a car with capacity empty seats. The vehicle only drives east (i.e., it cannot turn around and drive west).

You are given the integer capacity and an array trips where trips[i] = [numPassengersi, fromi, toi] indicates that the ith trip has numPassengersi passengers and the locations to pick them up and drop them off are fromi and toi respectively. The locations are given as the number of kilometers due east from the car's initial location.

Return true*if it is possible to pick up and drop off all passengers for all the given trips, or*false*otherwise*.

**Example 1:**

**Input:** trips = [[2,1,5],[3,3,7]], capacity = 4

**Output:** false

**My sol:**

class Solution:

def carPooling(self, trips: List[List[int]], capacity: int) -> bool:

dict={}

for x in trips:

i=x[1]

j=x[2]

for y in range(i,j):

if y in dict:

dict[y]+=x[0]

else:

dict[y]=x[0]

for k,v in dict.items():

if v>capacity:

return False

return True

**1099. Two Sum Less Than K**

Given an array nums of integers and integer k, return the maximum sum such that there exists i < j with nums[i] + nums[j] = sum and sum < k. If no i, j exist satisfying this equation, return -1.

**Example 1:**

**Input:** nums = [34,23,1,24,75,33,54,8], k = 60

**Output:** 58

**Explanation:** We can use 34 and 24 to sum 58 which is less than 60.

**My sol:**

class Solution:

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

max=-1

for x in range(0,len(nums)):

for y in range(0,len(nums)):

if x<y and nums[x]+ nums[y]<k and nums[x]+ nums[y]>max:

max=nums[x]+nums[y]

return max

**1100. Find K-Length Substrings With No Repeated Characters**

Given a string s and an integer k, return *the number of substrings in*s*of length*k*with no repeated characters*.

**Example 1:**

**Input:** s = "havefunonleetcode", k = 5

**Output:** 6

**Explanation:** There are 6 substrings they are: 'havef','avefu','vefun','efuno','etcod','tcode'.

**My sol:**

class Solution:

def numKLenSubstrNoRepeats(self, s: str, k: int) -> int:

i=0

count=0

if k==1:

return len(s)

while i+k<=len(s):

if len(list(set(s[i:i+k])))==k:

count+=1

i+=1

return count

1101-1150 ($$$$$ 11)

**1103. Distribute Candies to People**

We distribute some number of candies, to a row of **n = num\_people** people in the following way:

We then give 1 candy to the first person, 2 candies to the second person, and so on until we give n candies to the last person.

Then, we go back to the start of the row, giving n + 1 candies to the first person, n + 2 candies to the second person, and so on until we give 2 \* n candies to the last person.

This process repeats (with us giving one more candy each time, and moving to the start of the row after we reach the end) until we run out of candies.  The last person will receive all of our remaining candies (not necessarily one more than the previous gift).

Return an array (of length num\_people and sum candies) that represents the final distribution of candies.

**Example 1:**

**Input:** candies = 7, num\_people = 4

**Output:** [1,2,3,1]

**Explanation:**

On the first turn, ans[0] += 1, and the array is [1,0,0,0].

On the second turn, ans[1] += 2, and the array is [1,2,0,0].

On the third turn, ans[2] += 3, and the array is [1,2,3,0].

On the fourth turn, ans[3] += 1 (because there is only one candy left), and the final array is [1,2,3,1].

**My sol:**

class Solution:

def distributeCandies(self, candies: int, num\_people: int) -> List[int]:

i=1

arr=[0 for x in range(0,num\_people)]

while candies>0:

for x in range(0,len(arr)):

arr[x]+=i

candies-=i

if i+1<candies:

i+=1

else:

i=candies

if candies<=0:

return arr

**1108. Defanging an IP Address**

Given a valid (IPv4) IP address, return a defanged version of that IP address.

A *defanged IP address* replaces every period "." with "[.]".

**Example 1:**

**Input:** address = "1.1.1.1"

**Output:** "1[.]1[.]1[.]1"

**My sol:**

class Solution:

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

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

**1118. Number of Days in a Month**

Given a year year and a month month, return *the number of days of that month*.

**Example 1:**

**Input:** year = 1992, month = 7

**Output:** 31

**My sol:**

class Solution:

def numberOfDays(self, year: int, month: int) -> int:

if month ==1:

return 31

elif year==1700 or year==1800 or year==1900 or year==2100 and month==2:

return 28

elif year % 4 == 0 and month==2:

return 29

# if month ==2 and year %4 == 0 and year % 400 != 0:

# return 28

# if month ==2 and year %4 != 0:

# return 28

elif month ==3:

return 31

elif month ==4:

return 30

elif month ==5:

return 31

elif month ==6:

return 30

elif month ==7:

return 31

elif month ==8:

return 31

elif month ==9:

return 30

elif month ==10:

return 31

elif month ==11:

return 30

elif month ==12:

return 31

elif year % 4 == 0 and month==2 and year % 400 == 0:

return 29

else:

return 28

**1119. Remove Vowels from a String**

Given a string s, remove the vowels 'a', 'e', 'i', 'o', and 'u' from it, and return the new string.

**Example 1:**

**Input:** s = "leetcodeisacommunityforcoders"

**Output:** "ltcdscmmntyfrcdrs"

**My sol:**

class Solution:

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

for x in s:

if x=='a' or x=='e' or x=='i' or x=='o' or x=='u':

s=s.replace(x,"")

return s

**1122. Relative Sort Array**

Given two arrays arr1 and arr2, the elements of arr2 are distinct, and all elements in arr2 are also in arr1.

Sort the elements of arr1 such that the relative ordering of items in arr1 are the same as in arr2. Elements that do not appear in arr2 should be placed at the end of arr1 in **ascending** order.

**Example 1:**

**Input:** arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]

**Output:** [2,2,2,1,4,3,3,9,6,7,19]

**My sol:**

class Solution:

def relativeSortArray(self, arr1: List[int], arr2: List[int]) -> List[int]:

dup=[]

for i in arr1:

if i not in arr2:

dup.append(i);

arr=[]

dic={}

for i in arr1:

if i in dic:

dic[i]+=1;

else:

dic[i]=1;

for x in arr2:

for key,value in dic.items():

if x == key:

for i in range(0,value):

arr.append(x);

arr =arr + sorted(dup);

return arr

**1128. Number of Equivalent Domino Pairs**

Given a list of dominoes, dominoes[i] = [a, b] is **equivalent to** dominoes[j] = [c, d] if and only if either (a == c and b == d), or (a == d and b == c) - that is, one domino can be rotated to be equal to another domino.

Return *the number of pairs*(i, j)*for which*0 <= i < j < dominoes.length*, and*dominoes[i]*is****equivalent to***dominoes[j].

**Example 1:**

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

**Output:** 1

**My sol:**

class Solution:

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

dict={}

ans=0

for x in dominoes:

a,b=min(x),max(x)

if str(a)+str(b) not in dict:

dict[str(a)+str(b)]=1

else:

ans+=dict[str(a)+str(b)]

dict[str(a)+str(b)]+=1

return ans

**1133. Largest Unique Number**

Given an integer array nums, return *the largest integer that only occurs once*. If no integer occurs once, return -1.

**Example 1:**

**Input:** nums = [5,7,3,9,4,9,8,3,1]

**Output:** 8

**Explanation:** The maximum integer in the array is 9 but it is repeated. The number 8 occurs only once, so it is the answer.

**My sol:**

class Solution:

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

min=-1

dict={}

for x in nums:

if x in dict:

dict[x]+=1

else:

dict[x]=1

for key,value in dict.items():

if value==1 and key>min:

min=key

return min

**1134. Armstrong Number**

Given an integer n, return true *if and only if it is an****Armstrong number***.

The k-digit number n is an Armstrong number if and only if the kth power of each digit sums to n.

**Example 1:**

**Input:** n = 153

**Output:** true

**Explanation:** 153 is a 3-digit number, and 153 = 13 + 53 + 33.

**My sol:**

class Solution:

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

count=0

for x in str(n):

count+=pow(int(x),(len(str(n))))

return count==n

**1137. N-th Tribonacci Number**

The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given n, return the value of Tn.

**Example 1:**

**Input:** n = 4

**Output:** 4

**Explanation:**

T\_3 = 0 + 1 + 1 = 2

T\_4 = 1 + 1 + 2 = 4

**My sol:**

class Solution:

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

L=[0,1,1]

for i in range(3,n+1):

L.append(L[i-1]+L[i-2]+L[i-3])

print(L)

return L[n]

**1143. Longest Common Subsequence**

Given two strings text1 and text2, return *the length of their longest****common subsequence****.*If there is no **common subsequence**, return 0.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

* For example, "ace" is a subsequence of "abcde".

A **common subsequence** of two strings is a subsequence that is common to both strings.

**Example 1:**

**Input:** text1 = "abcde", text2 = "ace"

**Output:** 3

**Explanation:** The longest common subsequence is "ace" and its length is 3.

**My sol:**

class Solution:

def longestCommonSubsequence(self, text1: str, text2: str) -> int:

dp=[[0 for i in range(len(text2)+1)] for j in range(0,len(text1)+1)]

# print(dp)

for x in range(len(text1)-1,-1,-1):

for y in range(len(text2)-1,-1,-1):

if text1[x]==text2[y]:

dp[x][y]=1+dp[x+1][y+1]

else:

dp[x][y]=max(dp[x][y+1],dp[x+1][y])

# print(dp)

return dp[0][0]

**1150. Check If a Number Is Majority Element in a Sorted Array**

Given an integer array nums sorted in non-decreasing order and an integer target, return true *if* target *is a****majority****element, or*false*otherwise*.

A **majority** element in an array nums is an element that appears more than nums.length / 2 times in the array.

**Example 1:**

**Input:** nums = [2,4,5,5,5,5,5,6,6], target = 5

**Output:** true

**Explanation:** The value 5 appears 5 times and the length of the array is 9.

Thus, 5 is a majority element because 5 > 9/2 is true.

**My sol:**

class Solution:

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

if target in nums and nums.count(target)>(len(nums)/2):

return True

else:

return False

1151-1200 ($$$$$ 13)

**1154. Day of the Year**

Given a string date representing a [Gregorian calendar](https://en.wikipedia.org/wiki/Gregorian_calendar) date formatted as YYYY-MM-DD, return *the day number of the year*.

**Example 1:**

**Input:** date = "2019-01-09"

**Output:** 9

**Explanation:** Given date is the 9th day of the year in 2019.

**My sol:**

class Solution:

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

count=0;

d=date[8:]

m=date[5:7]

y=date[:4]

print(d,y,m)

flag=1

if int(y)%4==0:

if int(y)%100==0 and int(y)%400==0:

flag=0

elif int(y)%100==0 and int(y)%400!=0:

flag=1

else:

flag=0

if m=='01' :

count+=int(d)

elif m=='02':

count+=31+int(d)

elif m=='03' and flag==0:

count+=60+int(d)

elif m=='03' and flag!=0:

count+=59+int(d)

elif m=='04' and flag==0:

count+=91+int(d)

elif m=='04' and flag!=0:

count+=90+int(d)

elif m=='05' and flag==0:

count+=121+int(d)

elif m=='05' and flag!=0:

count+=120+int(d)

elif m=='06' and flag==0:

count+=152+int(d)

elif m=='06' and flag!=0:

count+=151+int(d)

elif m=='07' and flag==0:

count+=182+int(d)

elif m=='07' and flag!=0:

count+=181+int(d)

elif m=='08' and flag==0:

count+=213+int(d)

elif m=='08' and flag!=0:

count+=212+int(d)

elif m=='09' and flag==0:

count+=244+int(d)

elif m=='09' and flag!=0:

count+=243+int(d)

elif m=='10' and flag==0:

count+=274+int(d)

elif m=='10' and flag!=0:

count+=273+int(d)

elif m=='11' and flag==0:

count+=305+int(d)

elif m=='11' and flag!=0:

count+=304+int(d)

elif m=='12' and flag==0:

count+=335+int(d)

elif m=='12' and flag!=0:

count+=334+int(d)

return count

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

You are given an array of strings words and a string chars.

A string is **good** if it can be formed by characters from chars (each character can only be used once).

Return *the sum of lengths of all good strings in words*.

**Example 1:**

**Input:** words = ["cat","bt","hat","tree"], chars = "atach"

**Output:** 6

**Explanation:** The strings that can be formed are "cat" and "hat" so the answer is 3 + 3 = 6.

**My sol:**

class Solution:

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

final=""

dup=chars

flag=0

for w in words:

for y in w:

if y in chars:

chars=chars.replace(y,"",1)

flag=0

else:

flag=1

break

chars=dup

if flag==0:

final=final+w

return len(final)

**1161. Maximum Level Sum of a Binary Tree**

Given the root of a binary tree, the level of its root is 1, the level of its children is 2, and so on.

Return the **smallest** level x such that the sum of all the values of nodes at level x is **maximal**.

**Example 1:**

![Shape, arrow

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDsRXhpZgAATU0AKgAAAAgABAE7AAIAAAALAAAISodpAAQAAAABAAAIVpydAAEAAAAWAAAQzuocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGtoYWxlZCBhbGkAAAAFkAMAAgAAABQAABCkkAQAAgAAABQAABC4kpEAAgAAAAM4MgAAkpIAAgAAAAM4MgAA6hwABwAACAwAAAiYAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAxOTowNTowNCAwMDowNzozNgAyMDE5OjA1OjA0IDAwOjA3OjM2AAAAawBoAGEAbABlAGQAIABhAGwAaQAAAP/hCx1odHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDE5LTA1LTA0VDAwOjA3OjM2LjgyMjwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5raGFsZWQgYWxpPC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIASMBTAMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APpGiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKR3WONnkYIiglmY4AHqa5Y+INU8QyNF4ShjjtFYq+qXSnYT/0zTq/1PFNK4m7HVU0SIW2hlLemea5oeB7a7+bX9R1DVnP3hLOY4/wRMAU8/D7wqV2/wBjwj3DuD+ec0aCu+x0lFcwfBr2Pz+HNZvtOYfdhkkM8J9ij5/MGltfE11pt5Fp/i62js5ZW2wXsJJt5z6ZPKN7Gi3YL9zpqKKKRQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFZfiTVDovhq+1BeXhiPljHVzwv/AI8RQGxjXm/xhr02loxXRNPcLespx9qm6+Vn+6v8XvxXVxxpDEscKLHGgCqijAUegFZvhrSRonh60sjzKqbpnPV5G5Yk/UmtSmyUurCiiqw1GzOpnT1uEa7EfmmEHLKucZPp170iizVe+sbbUrKW0voVnglXa6OOD/n1qxRQBy+g3NzomsHw1qkzTIUMmm3LnLSxDrGx/vL+orqK5zxvayNoH9pWg/0zSpBeQn/d+8v0K5retbmO8s4bmE5jmjWRD6gjI/nTfclaaEtFFFIoKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK5nx18+k2EDfcn1O2jf3BkB/pXTVznjyJz4SnuYV3SWMkd2oH+w4J/TNOO5Mtjo6KZDMlxbxzQtujkUOrDuCMg0+kUMni8+3ki3vH5ild8Zwy5HUHsa47SNFsdD+Ixt9PiKK2l75HdizyN5nLMx5JrtK577NP/wsj7T5Mn2f+zNnm7Ds3eZnbnpnHagDoaKKKAIL6JZ9PuIn+7JEyn6EEVj+BpWl8C6Qz9RbKv4DgfoKt+Jb9dM8L6jeOceVbvt92Iwo/Mil8N2Lab4Y02zkGHhtkVx6NtGf1zT6E/aNOiiikUFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFMmhjuIJIZlDxyKUdT0IIwRT6KAOX8JXL6bJP4Xv3/ANI0/m1Zus9sT8jD6fdPpgV1Fc142trVdKXVDO1pqFid1nPGuXLn/lnj+IN0IpNH8XCWSKw8SW50fVGUERTHEc3ujdD9M5HTmqavqQnZ2Z01FFFSWFFIzBFLMQqgZJJ4Arjdf8ZzS2tzB4Rge/kgB+0Xsa7orYdyOztjnAppNibSLGtP/wAJH4lttCt/ns7J1utRcdMjmOI+5PJHoK6usfwxp+n2GhxHS5ftKXH76S6Y5ad26ux9fbt0rYofYS7hRRRSKCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACo7i4htLaS4uZFihiUs7scBQO9SVybH/hNNTaIZ/sGyk+c9r2UdvdFP5mmlcTdiXSbabxFqqa/qKMlnFn+zbVx0H/PZh/ePb0Fb2oabZaram21G1iuYW/gkXOPceh96sgAAADAHQCihsErI5geDprH/AJAGv6jpyD7sDsLiJfor5I/Onf2L4rPB8Wxgf3hpaZ/9CxXS0UXYuVHMjwVFeMG8Qapf6vg58mWXy4c/7iYH510NtawWdukFpDHBCgwscahVUewFS0UXbGkkcpID4M1Rpxn+wb2T94vaylP8Q/2GPX0NdUCGUFSCCMgjvTLi3iu7eS3uY1likUq6MMhge1c3pVxL4b1WPQb9meym/wCQbcuf/ILH1Hb1FPcn4X5HUUUUVJYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRWF4g1ieKaLSNHw+q3gOwnlbdO8rew7epppXE3YrazdXGu6k3h/SZTHEuDqN0n/LJD/yzU/3m/QV0NrawWNpFbWkaxQxKFRFHAFVdF0e30PTUtLbc3JeSVzlpXPVmPqav0N9EJLqwooopFBRRRQAUUUUAFU9W0u31nTZLO7B2PyrL95GHRgexFXKKA3MDw/qlytw+ia4w/tK2XKS4wLqLtIPf1HrW/WTr+if2vbxy20n2fULVvMtbgfwN6H1U9CKNA1saxbSJPF9nv7VvLurc9Y39R6qeoNU9dSVpozWoooqSgooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooqrqWpW2k6fLe30nlwxDJPcnsAO5NAFXXtaXRrEOkZuLuZvLtbZfvSueg+nqe1R+H9GfTopbrUHE+qXhD3U3v2RfRR0FVdB025ur5tf1pSt3Mu22tzyLSI9v8AePc/hXRVT00JWruwoooqSgooooAKKKKACiiigAooooAK5/xBpl1HcJrmhoDqNuu2SLoLqLuh9+4PrXQUU07CauU9K1S21jTYr2zbMbjlTwyN3UjsRVyuX1SCTwzqkuvWKM9hOf8AiZWydv8Apso9R3HcV0kE8V1bxz28iyRSKGR1OQwPQ0NdUJPoySiiikUFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVz/ibVbqOS30XRGA1S/B2yEZFvEPvSn6dAO5prUTdh2reJ1tb7+zNItX1TVCMmCI4WEf3pH6KP1qqvh/XdU+fXtfmtlbn7LpX7pV9vMOWP6VraHoVnoGni2s1LMx3zTOcvM56sx7k1pU722Fa+5zP/Cv9BbmaO7mf+/JezE/o1B8GG1+bRNd1SwcfdVp/Pi/FHzn866ailzMOVHK/wDCQat4edU8V2yS2ZOBqdmpKL/10Tqv1GRRp6N4u1KPV7tCNJtXJsIGH+uYcecw9P7o/GupdFkRkkUMrDDKwyCPSuQRD4H1iJEZj4f1CXYqE5FjM3THojfof1pPsJq2+x2FFFFQWFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAIyhlKsAQRggjrXJG4XwLfeXcEr4fumJjfk/Y5DyV9dh7ehrrJJEhieSVgiIpZmJwAB1NcjplofGOoprupq39lwsf7Ns3HDY485x3J7A9BVLzJl5bkqal4h8R/NosK6Pp7fdvLyPfNIPVI+gHu1SjwPbXHzavquq6i56+bdsifgqYArpqKV+wcvc5n/hANETm0+3Wj9ngvpQR+bGmtpXiXRvn0jVv7WhXraakBvI9pR3+oxXUUUXYcq6GPoniS21iSS2kikstQh/11lcDEi+4/vL7itisbxD4ej1qGOa3kNpqVqd1peJ96NvQ+qnuKXw3rbaxYyLdxeRqFpIYLyD+447j/ZPUGjzQJu9mbFFFFIoKKKKACiiigAooooAKKKKACiiigArl/CC/wBp3ep+I5fmN7O0NsT/AAwRnauPqQSa3dVla30a9mTho7eRh9QpNZ/gyFYPBOjonANpG/4su4/qafQnqbdIzBELNwFGSaWikUct/wALI8Lf9BCb/wAAZ/8A4it3StWstb06O+0yUzW0hIVyjJnBweGAPUVn+K72aDTY7GxfZe6lKLaFh1TP3n/Bcn8q1LCyh03T4LO1XZDAgRB7CgCxVPVtNg1jSLnT7oZiuIyh9vQ/UHB/CrlFAGF4O1CbUPDUH205vLVmtbjP/PSM7SfxwD+Nbtcz4W/da94mt1+4t+JQPd41J/lXTU3uTHYKKKKRQUUUUAFFFFABRRRQAUUUUAFFFFAHM+NHe7t7DQoWKvq1yIpCvUQqN0hH4DH410kUSQQpFCoSONQqqOgA4Arm7z998TtNRulvp00qexZ1U/pXTU3sSt2FVr/ULTS7N7vULiO3gT7zucD6e59qs1zGslbvx9oVjMN0McM93tPQuu1VOPbcTSKNTSfEWla20iaZdiWSLBeNkZHUHoSrAHHvitOuX8QBbXxn4cvIlxLPLLayMOrIYy2D64K11FABXL6mv9jeOtN1KP5YdUBsbr0LgFom+vBX6V1Fcz49+Tw7Fcj79tfW0qH0IlUf1prcmW1zpqKKKRQUUUUAFFFFABRRRQAUUUUAFFFFAEV1ALqzmt24EsbIfxGKw/AlwZ/BdhHJxLbKbaRf7rRkrj9BXQ1yls//AAjfjae1l+Ww1x/Pt27Jc4+dD/vDBHvxTW1iXo7nV0UUUijn/s8958QPPmglW10+yxDIykI8kjfNg9CQqgfjXQUUUAFFFYXizVpdP0xbTT/m1PUG+z2aZ/iPV/YKOc/SmtRN2RW8GH7U2tamOUvNSk8pv7yIAgP/AI6a6aqWjaZFo2i2mnW/KW8YTP8AePc/icn8au0PcFogooopDCiiigAooooAKKKKACiiigAooooA5nWD9i8faDeNxHcxz2bt6EgOg/Eqa6asfxTpMur6E8dm2y9gdbi0f+7Khyv59PxqbQNZi17Ror2IFHOUmiIwYpBwykex/pT6ErR2NKud8RWt1b6tpuu2Fq949lvingixveJwMlQepBAOK6KikUcvAbjxJ4jsb5tPurKw00O6G8i8t5ZWG3hTyAATyeua6iiigArmfGp+0Q6Tpi8ve6lCCv8AsId7H8AtdNXKaS//AAkfi6bWl+bT9OVrSybtJIf9bIPbgKD35pruTLsdXRRRSKCiiigAooooAKKKKACiiigAooooAKo6xpFtrmlyWN4DsflXU4aNh0ZT2INXqM460AcpY+IbnQZ4tK8YOEdjtttTxiK5Hbcf4H9QeK6pWDKGUgqRkEHrXKWVvD4v1mbU72JJ9Ktg1vZRSLuSY9HlIPX+6Kl/4Q1rBifDes3mkrnPkZE8A+iP0/A1TSITZ09Fcz9i8Zx/Kur6VKP78lmyn8g2KQ+HtfvuNV8TypEesWn26wH/AL7OWpW8x3fYva34nstGdLf5rvUJuIbG3+aRz2yP4R7moNC0S6W+k1rX2SXVJl2JGhylpH/zzT39T3/nXu/BFhBpDLocK2+oxOJ4bpmLSNKvI3OeSDyCOnNa+g6umtaTHdBfLlBMc8R6xSLwyn8f0xT6aC1vqaNFFFSWFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFcxqml32katJrnh2MTGXH27T84FwB/Gno4/X69enopp2E1czdG8Qadr0BewmzIhxLBINssR9GU8j+VaVY+reFtL1idbmeJ4LxfuXds5imX/gQ6/jmqI0TxPZ8WHiZbiMfdS/tFdh9XUgn8qNBXa3Ompk00VvC01xIkUSDLO7BVUe5Nc59g8ZTfLJrWm2w/vwWRc/kzYpY/BVtcTLP4hvrvWpFOQly2IVPqIlwv55osgu+xVuNRuvGbNY6E0lvpGdt1qRG0zDukWeuehb/J6mzs4NPs4rSziWKCFQiIo4AFSoixoqRqFVRhVUYAHpS0NjSCiiikMKKKKACiiigAooooAKKKKACiiigArnfE93NdzQeHtOcpc34JnkXrBbj7zfU/dH1rY1LUINK02e+vG2wwIWY+voB7k8Vl+GNPuFin1fVFxqOokSOp58mP8AgjH0HX3qlpqS9dDYtLWGxs4rW1QRwwoERR2AqaiipKCiiigArl9Q/wCKa8Srqq/LpupMsN6O0UvRJPoehP411FQX1lBqNjNZ3ab4ZkKOvsaadhNXJ6K57wvezwmfQdTcte6dgK5/5bwn7j/XsfcV0NDVgTugooopDCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKxvEuqy6dYJBYAPqN6/kWiH+8erH2UcmmtRN2Vyhcf8VN4pFoPm0zSXDz+k1x/CnuF6n34rqKoaJpMWi6TDZQkvsGZJG6yOeWY+5NX6GxJdwooopFBRRRQAUUUUAc/4osZ0EGuaYm6+07LFB/y2hP30PrxyPetiwvoNS0+C9tH3wzoHQ+x/rViuXsf+KZ8TNprfLpupu0toe0U3V4/YHqPyqt0Ts7nUUUUVJQUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUVhat4ois73+zdLtpNU1QjP2aAgCP3kfog+vNVRo/iXVPn1fXBp0bf8u2lxgFfrI2Tn6DFOxN+x09Fcz/AMIPanl9Y1x3/vtqD5pD4Z1ix+bRvE96SP8AllqIW4RvbOAw/OiyC77HSySJDE8krBERSzMTwAOprm/D0b61qc3iW6UiN1MOnxsPuQ55fHYsf0rG1vV9Unez0PxPax6ZbXUwW4v4ZCYJkHOxW6oWPHzdq72NEjiRIlCxqoCqvQAdAKeyF8THUUUVJYUUUUAFFFFABRRRQAVna7pEet6TLaOxjk4eGUdYpByrD6GtGijYNzI8N6vJqumst4ojv7VzBdx9MOO49j1Fa9cb4jv10HxZZ3umKbu+vF8m50+HmSZAPlk9tp4ye30q0LHxXrHz3+ow6JC3/LtZIJZcejSNwD/uiqa6kJ9DqKK5n/hCLduZ9a1yZv7z6gw/QYFB8KX9p82keJ9UicdFvGW5T6YYZ/WlZDu+x01Fct/wkep6E6x+LbJBbEhV1Kyy0Q/31PzJ9eRXTxSxzRLLC6yRuNyuhyGHqDQ1Yadx1FFFIYUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABXO+JNVvPtEGhaE6rql4pYykZFrCOGkPv2A9a6FmCqWYgADJJ7VzHguM38V74inB87VJiYt3VIFO2NfbgZ98013JfY1tD0Kz0Cw+zWSksx3TTOcvM/dmPc1pUU2SRIo2klZURQSzMcAD1JpFbDqKx/8AhLvDf/QwaV/4Gx//ABVadtcwXlulxaTxzwyDKSROGVh7EcGgAurWC+tZLa8iSaCVdrxuMhhXMaZJN4T1mHRLyZ5tLvCRp08hy0Tf88WP/oJ/CusrK8SaQNc0C5s1O2bbvgcHBSVeVYH6imuxLXVGrRWX4a1U634bsr9xiSWPEox0cHaw/MGtSkUtQooooAKKKKACiiigArF8S61JpVrDb6fGJ9Tvn8mzhPQt3Y/7Kjk1tVy+hL/bHizVtbl+aO1c6fZg9FC8yMPqxxn0FNdyX2L/AIe8OxaJDJLLIbvUrk7rq8k+9K3oPRR2FbNFFLcaVgoozjrVS11XTr6aSGyv7W5lj++kMyuyfUA8UDLToskbJIodGBDKwyCPQ1x/lnwLqiNGzHw7eShGjJz9hlY8Ef8ATMnt2NdjVbUbCDVNNuLG7XdDcRmNx7Hv9aaYmrlmiuf8GXs9xoP2S+bdd6bM9lM394ocBvxXaa6Ch6AndXCiiikMKKKKACiiigAooooAKKKKACiiigAooooAKKKKAMrxTO1t4R1aVOGWzlKn0Ow4qXw/Atr4a02BBhY7WJR/3yKXXbNtQ8PajZoMvPbSRqPcqQP1qv4TvFv/AAjpdwpzutUVv95Rtb9QafQn7Rr0joskbJIodGGGVhkEelLRSKOS8U6ZpkFhFY6fpVgl9qMot4WFsmUB5Z+n8K5P1xXS6fYwaZp0FlaIEhgQIij0FYdv/wATL4hXcrcxaVarDHz/AMtJPmY/98hR+NdJQAUUUdOtAHM+Cf3cOt2y/ct9YuEQeina2P8Ax6umrmfAh8/R73UB9zUdRuLlPdS20fotdNTluTHYKKKKRQUUUUAFFFFACO2yNmP8IzXOfD5MeBdPduXmDyufVmdif510hAIIPIPWuZ8AMY/CaWL/AOt0+ea1kHoVc/0Ip9CftHTUUVnato0errGJLy/tfLzj7HdPDuz67TzSKKnjK3nuvCN/FbSBG8vLZcIGQEFl3HgZGRmuWF7oGs6loMPha3S1vIplk8wQGDZCB86gkDzM9MLu9a1/FGjT2vhOKCz+2ajFb3cc88U0zTSyxBssuTy3rj2xVPW/EWneJ7S203w4ZbrUPtMTrshdPsu1gSzEgbcDIx36UAd1RRRQBzOhfuvHPieBeEY20wHoWjIP/oNdNXM+Gj9q8UeJtQXmNrqO1U+8SYb9WNdNTe5MdgooopFBRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVyugv8A2B4lvfD83ywXTte6cexVuZIx/unnHoc11VZev6HHrlgsYla3uoXEtrcp96GQdCPb1HcU0J90alFc5pHiZ1ul0jxKi2OqjhSTiK6H96Nv/Zeoro6LWBO4UUUUhhXPeMNSlg01NL0451LVGNvbqOqg/fk+irk5+lWdb8S2Wi7IW3XV/NxBZQDdJIfp2Huag0HRbpLyTWteZJNVuF2BEOUtY+ojT+p7n9WtNWS3fRGtpthFpel21jbDEVvEsa+4AxmrNFFIoKKKKACiiigAooooAK5UP/wjnjpxJ8un67go3aO6UYx/wNf1FdVVLV9Ktta0yWxvVJjkHDLwyMOjKexBpoTRdorldP1+50O4j0nxc4RydtrqRGIrkdgx/hf1B611QIIBByD0NDVgTuFFFFIYVleJNaXQ9FkuVXzLlyIrWEdZZW4VR+P6CpNZ17T9BthLqE21nOI4UG6SU+ir1JrL0nS73VNVTXvEUXlPGCLGxzkWynqzesh/T+TS6slvojQ8NaQdE8P21nK2+fBknf8AvyMdzH8zWrRRSKWgUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQBU1LSrHWLNrXU7WO5hb+Fx09weoPuKwl8NavpnHh/xDMkI6WuoRidB7BuGA/E11FFO7E0mcz/AMVyOMeHn/2szr+mD/OkOj+KdR41PX4bKI/ej0y3wx+juSR+VdPRRcXKZej+HNN0Pe1lATPJ/rbmVi8sn1Y8/wBK1KKKRWwUUUUAFFFFABRRRQAUUUUAFFFFAEN3aW9/avbXsEc8Mgw0cihgfwrnR4Su9MP/ABTGt3FhF2tJ1FxCPYBuVH0NdRRTu0JpM5nHjlPlB0CYf3285CfwGaQ6d4vvvlvNasdPQ9RYWxdiP96Q8fXFdPRRcXKYuk+FdO0m5N2BLd3zDDXl2/mSn6E9PwxW1RRS3GklsFFFFAwooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA/9k=)

**Input:** root = [1,7,0,7,-8,null,null]

**Output:** 2

**Explanation:**

Level 1 sum = 1.

Level 2 sum = 7 + 0 = 7.

Level 3 sum = 7 + -8 = -1.

So we return the level with the maximum sum which is level 2.

**My sol:**

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

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

res=[]

q=collections.deque()

q.append(root)

while q:

qlen=len(q)

level=[]

for x in range(qlen):

node=q.popleft()

if node:

level.append(node.val)

q.append(node.left)

q.append(node.right)

if level:

res.append(sum(level))

print(res)

return res.index(max(res))+1

**1165. Single-Row Keyboard**

There is a special keyboard with **all keys in a single row**.

Given a string keyboard of length 26 indicating the layout of the keyboard (indexed from 0 to 25). Initially, your finger is at index 0. To type a character, you have to move your finger to the index of the desired character. The time taken to move your finger from index i to index j is |i - j|.

You want to type a string word. Write a function to calculate how much time it takes to type it with one finger.

**Example 1:**

**Input:** keyboard = "abcdefghijklmnopqrstuvwxyz", word = "cba"

**Output:** 4

**Explanation:** The index moves from 0 to 2 to write 'c' then to 1 to write 'b' then to 0 again to write 'a'.

Total time = 2 + 1 + 1 = 4.

**My sol:**

class Solution:

def calculateTime(self, keyboard: str, word: str) -> int:

a=[0]

count=0

for x in word:

a.append(keyboard.index(x))

print(a)

for x in range(1,len(a)):

print(abs(a[x]-a[x-1]))

count+=abs(a[x]-a[x-1])

return count

**1167. Minimum Cost to Connect Sticks**

You have some number of sticks with positive integer lengths. These lengths are given as an array sticks, where sticks[i] is the length of the ith stick.

You can connect any two sticks of lengths x and y into one stick by paying a cost of x + y. You must connect all the sticks until there is only one stick remaining.

Return *the minimum cost of connecting all the given sticks into one stick in this way*.

**Example 1:**

**Input:** sticks = [2,4,3]

**Output:** 14

**Explanation:** You start with sticks = [2,4,3].

1. Combine sticks 2 and 3 for a cost of 2 + 3 = 5. Now you have sticks = [5,4].

2. Combine sticks 5 and 4 for a cost of 5 + 4 = 9. Now you have sticks = [9].

There is only one stick left, so you are done. The total cost is 5 + 9 = 14.

**My sol:**

class Solution:

def connectSticks(self, sticks: List[int]) -> int:

a=sorted(sticks)

x=1

temp=0

res=0

while x<len(a):

temp=0

temp+=a[x]+a[x-1]

a[x]=a[x]+a[x-1]

res+=temp

a.pop(0)

a=sorted(a)

return res

**1170. Compare Strings by Frequency of the Smallest Character**

Let the function f(s) be the **frequency of the lexicographically smallest character** in a non-empty string s. For example, if s = "dcce" then f(s) = 2 because the lexicographically smallest character is 'c', which has a frequency of 2.

You are given an array of strings words and another array of query strings queries. For each query queries[i], count the **number of words** in words such that f(queries[i]) < f(W) for each W in words.

Return *an integer array*answer*, where each*answer[i]*is the answer to the*ith*query*.

**Example 1:**

**Input:** queries = ["cbd"], words = ["zaaaz"]

**Output:** [1]

**Explanation:** On the first query we have f("cbd") = 1, f("zaaaz") = 3 so f("cbd") < f("zaaaz").

**My sol:**

class Solution:

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

dummy1=[]

dummy2=[]

for x in words:

dummy1.append(x.count(sorted(x)[0]))

for x in queries:

dummy2.append(x.count(sorted(x)[0]))

count=0

ans=[]

for x in dummy2:

for y in dummy1:

if x<y:

count+=1

ans.append(count)

count=0

return ans

**1176. Diet Plan Performance**

A dieter consumes calories[i] calories on the i-th day.

Given an integer k, for **every** consecutive sequence of k days (calories[i], calories[i+1], ..., calories[i+k-1] for all 0 <= i <= n-k), they look at *T*, the total calories consumed during that sequence of k days (calories[i] + calories[i+1] + ... + calories[i+k-1]):

* If T < lower, they performed poorly on their diet and lose 1 point;
* If T > upper, they performed well on their diet and gain 1 point;
* Otherwise, they performed normally and there is no change in points.

Initially, the dieter has zero points. Return the total number of points the dieter has after dieting for calories.length days.

Note that the total points can be negative.

**Example 1:**

**Input:** calories = [1,2,3,4,5], k = 1, lower = 3, upper = 3

**Output:** 0

**Explanation**: Since k = 1, we consider each element of the array separately and compare it to lower and upper.

calories[0] and calories[1] are less than lower so 2 points are lost.

calories[3] and calories[4] are greater than upper so 2 points are gained.

**My sol:**

class Solution:

def dietPlanPerformance(self, calories: List[int], k: int, lower: int, upper: int) -> int:

i=0

temp=sum(calories[i:i+k-1])

res=0

while i+k<=len(calories):

temp+=calories[i+k-1]

if temp>upper:

res+=1

elif temp<lower:

res-=1

temp-=calories[i]

i+=1

return res

**1180. Count Substrings with Only One Distinct Letter**

Given a string s, return *the number of substrings that have only****one distinct****letter*.

**Example 1:**

**Input:** s = "aaaba"

**Output:** 8

**Explanation:** The substrings with one distinct letter are "aaa", "aa", "a", "b".

"aaa" occurs 1 time.

"aa" occurs 2 times.

"a" occurs 4 times.

"b" occurs 1 time.

So the answer is 1 + 2 + 4 + 1 = 8.

**My sol:**

class Solution:

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

count=0

a=list(s[i:j+1] for i in range (len(s)) for j in range(i,len(s)))

for x in a:

if x.count(x[0])==len(x):

count+=1

return count

**1185. Day of the Week ( import date)**

Given a date, return the corresponding day of the week for that date.

The input is given as three integers representing the day, month and year respectively.

Return the answer as one of the following values {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}.

**Example 1:**

**Input:** day = 31, month = 8, year = 2019

**Output:** "Saturday"

**My sol:**

import datetime

from datetime import date

import calendar

class Solution:

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

dat = datetime.date(year, month, day)

print(dat)

w=dat.weekday()

if w+1==1:

return "Monday"

elif w+1==2:

return "Tuesday"

elif w+1==3:

return "Wednesday"

elif w+1==4:

return "Thursday"

elif w+1==5:

return "Friday"

elif w+1==6:

return "Saturday"

elif w+1==7:

return "Sunday"

else:

return "invalid day"

**1189. Maximum Number of Balloons**

Given a string text, you want to use the characters of text to form as many instances of the word **"balloon"** as possible.

You can use each character in text **at most once**. Return the maximum number of instances that can be formed.

**Example 1:**

**![A picture containing text, clipart

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDsRXhpZgAATU0AKgAAAAgABAE7AAIAAAALAAAISodpAAQAAAABAAAIVpydAAEAAAAWAAAQzuocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGtoYWxlZCBhbGkAAAAFkAMAAgAAABQAABCkkAQAAgAAABQAABC4kpEAAgAAAAMzNgAAkpIAAgAAAAMzNgAA6hwABwAACAwAAAiYAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAxOTowOTowNSAxMjo1NDoxMwAyMDE5OjA5OjA1IDEyOjU0OjEzAAAAawBoAGEAbABlAGQAIABhAGwAaQAAAP/hCx1odHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvADw/eHBhY2tldCBiZWdpbj0n77u/JyBpZD0nVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkJz8+DQo8eDp4bXBtZXRhIHhtbG5zOng9ImFkb2JlOm5zOm1ldGEvIj48cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIi8+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczp4bXA9Imh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iPjx4bXA6Q3JlYXRlRGF0ZT4yMDE5LTA5LTA1VDEyOjU0OjEzLjM2MDwveG1wOkNyZWF0ZURhdGU+PC9yZGY6RGVzY3JpcHRpb24+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iPjxkYzpjcmVhdG9yPjxyZGY6U2VxIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpsaT5raGFsZWQgYWxpPC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAD8BFQMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APpGiiigAooooAKp6rqtloelzajqk4t7SAAySFS23JAHABJ5IFXK4z4uf8ks1f8A7Y/+j0rOrJwpykuiOvA0I4jFUqMtpSSfzaRvaB4n0fxPbSz6FeC6jibY58tkKnGejAGtWvJPgF/yBdX/AOvhP/QTXrdRh6jq0lN9TpzfB08DjqmHpttR777J+QUUUVueWFFFFABRRRQAVW1HULXStPmvtQlENtAu6SQgnaPoOas1zHxJ/wCSb61/17/1FRUk4wcl0OnC0o1sRTpS2k0vvdi/4f8AFmieKUnbQb4XYtyol/duhXOccMB1wfyrYrxn9n3/AJmH/t2/9q17NWWHqOrSU5bs7c5wVPAY+phqTbjG2++qT8u4UUUV0HkhRRRQAUUUUAZOv+J9H8MW8U+u3gtY5mKofLd9x+igmrelarZa3pcOo6XOLi0nBMcgUjOCQeCARyCOa8v+Pv8AyBdI/wCvh/8A0EV1Hwj/AOSWaR/22/8AR71yRrSeIdLolf8AI9+tltKnk9PHpvmlLla0tb3vK/TuaHi7x/4Z8Cx2r+KtUFgLwsIP3Mkhfbjdwik8bh19a2NK1Wy1zSbbU9KuFubO6jEkMqggOp6HB5H0NfPH7XP/ADKP/b7/AO0K9d+Dv/JHPDP/AF4r/M11ngHa0UUUAFFFFABRRRQAUUUUAFcD45+KCeCdfttNfSWvBNAs7Si42bQWZcAbTk/KT1Fd9Xz/APHj/kerP/sGp/6NlrkxlSVKlzQep9Fw5gqGOx6o4iN42fVr8juvGnxcs/D182l6LbDU9RU7X+b93E393jlj7DGPXtXJ2/x0160vlTW9FtPLyNyRLJE4Hr8xP8q7z4c+BtP8NaLb3zKtzqd1EsktywyV3DO1fQc9ep/QSfFPQLbWvAl/NJCpubGI3MMuBuTbywz6FQePp6VjOOJlB1Oa3kejh62S08RHBOhzxb5XNtp32ul0XzvY6DQNesfEmjQ6npcm+CUdCMMjDqpHYiue+Ln/ACSzV/8Atj/6PSuM+AN7IRrVixJiHlTKM9CdwP54X8q7P4uf8ks1f/tj/wCj0rRVXVwrm+zOOWBWAz+nh4u6U4W9G0/1OX+AX/IF1f8A6+E/9BNd74v8X2Hg3R1vtRSWXzH8uKKIcu2Ce/AHHWuC+AX/ACBdX/6+E/8AQTXpOteHtK8RQQw61ZpdxwyebGrkgBsEdiM8HoeKWG5nhVyb/wDBKzt0I55UeITcLq6W/wAKPIZvjlr97Mw0XQbYKD0kEkxA99pWlsPjtqtteCPX9FtygOHFvujdfwYnP04r2m1tLaxt1gsreK3hX7scKBFH0A4rgvjRo1peeA59SkiT7VYvG0cuPm2s4Qrn0+bOPas6lPEQg5+0u15HZg8blGKxEMK8Ioxk0k+Zt66L+rnbaRq1nrukwajpkwmtp13I2MEeoI7EHjFWbi4htLWW4upUhghQvJJIwVUUDJJJ6ACvMPgNdyS+FtQtXJKQXe5M9tyjI/T9a5v9qXxZPpfhPTvDtnK0batK0lwV7xR4+U+xZgf+A12UKntKan3Pm80wawWNqYdO6i9PTdFLxT+1BjVm03wBoX9qNu2R3VzuxKf9iJfmI9yQfasiP9pXxvol3F/wmHg63igkPCrDNauR3wZCwP5V6H+z94DsPDXw8sdZe3RtW1iIXElwygssTcoinsNu0kdyfYV6Pr+gab4n0O50nWrVLmzuUKujDp6MD2YdQexrY84zfA/jrRfiB4eTVtBmJUHZNBIMSQPjO1h/UcGo/iT/AMk31r/r3/qK+bfgRf3Xg349XXhh5WMNzJcafODwC8O4q2PXKEf8CNfSXxJ/5JvrX/Xv/UVlW/hy9Gd2Xf77R/xR/NHk/wAJPFWmeEtL8Q3urS7QxtliiQZeVv3vCj+vQVduvjh4imZ7jTNCtUs1J+aZJJMD3YFQPyrC+FHgm28Xa1cTak2bLTwjSQjgzM2dq5/u/Kc/l3r6LitoILZbeCGOOBV2rEiAKB6ADjFebhYVqlFJS5UvxPt8+xOWYTMJyqUfa1JWvd2UVZJJd21qeeeBvi7aeJr5NM1a2WwvpDiFlbMcp/ujPKn0HOfXtXo9fOPxW8PReFfG8dxpK/ZobpBcxLHwIpA3IX05AP41734f1Uav4Y0/VHIX7RbJK5PAUlct+RzXTha03KVKpujw88y/DU6VHHYNWp1OnZ/1f7jN8Y+OtK8GWaPflprqUEw2sR+d/c+g9/yzXmL/ABv8S3kzHSNDtDGD91o5JiB7lSP5VzljHL8TfisPtkjfZ7mZnbBxsgTJCj04AH1Oa+jrHT7TTLNLTTraO2t4xhY4lCgf59ayhKtiW3CXLFHdiKGX5HThTxFL2taSu7uyXl/Xr5HkWifHd/tiw+JdKSKMnDTWhOU+qNnP5/nXr1leW+o2UN3YzLPbzIHjkQ5DA1wPxe8IWeq+FrnWIIETULFfMMirgyRj7wb1wOR9KxfgNrss1nqOiTNuS3IuIAewY4YfTOD+Jq6dSpTrexqO99mYY3BYLG5a8xwUORwdpRvdeq+9fiSfH3/kC6R/18P/AOgiuo+Ef/JLNI/7bf8Ao965f4+/8gXSP+vh/wD0EV1Hwj/5JZpH/bb/ANHvSh/vsvT/ACKxX/JMUP8Ar4//AG88g/a5/wCZR/7ff/aFeu/B3/kjnhn/AK8V/ma8i/a5/wCZR/7ff/aFaS/Fyz+HHwC8M29mYrnX7rT1+zWzHIiGSPNcf3R2H8R9gSPRPjTtPil8bNI+Gs8FgLQ6tqsq72tI5xGIk7F2w2Cewx78cZ6T4eeLb3xv4Qh1y/0OXRPtDnyYJZvMMkfGJAdqnB5xkds9CDXhPwd+Dt74z1T/AITn4iCS4t7iT7RBb3HLXrE58xx/zz9B/F/u9fp1VCqFUAADAAHSgDE8VeLtL8IaaLvVZDufIhgj5eU+gHp79BXlU/xy1+8uXXRNDtdgOQsiyTNj32la5/xLc3HxA+LZsUmIga6+xwEdEiU4LAe+Gb8a+g9H0XT9B02Ox0q2S3gjAGFHLH1Y9z7mvNjOriZvklyxX4n21TDYDJcPTeJpe1rTV7N2UV/X67Hkel/He9hvBF4j0aMJnDta7kdPfaxOfpkV69pOrWWuaXDqGlzrPbTDKuP1BHYg9q5n4k+ELPxL4Wu5hAg1G1iaW3mAwx2jJQnuCBj64rz/AOBGuyRaxfaHK/7ieL7REp7OpAOPqD/47ThUq0aypVHdPZmeIweBzHLp47Bw9nOn8Ub3Vu6/pbPTqe5UUUV6J8aFfP8A8eP+R6s/+wan/o2WvoCvn/48f8j1Z/8AYNT/ANGy1wZh/APreEP+Rov8LPctE/5F/T/+vWL/ANAFVPGX/Ii69/2Dbj/0U1W9E/5F/T/+vWL/ANAFVPGX/Ii69/2Dbj/0U1dUv4b9DwKX++x/xL8zyr4Af8hPWv8ArjF/Nq7v4uf8ks1f/tj/AOj0rhPgB/yE9a/64xfzau7+Ln/JLNX/AO2P/o9K4aH+5P0f6n1ea/8AJTw/x0//AG05f4Bf8gXV/wDr4T/0E11fxE8dp4J0iJoYlnv7olbeNz8ox1du+Bkcd81ynwC/5Aur/wDXwn/oJrI+PltMNc0q6IPkPbtGp7Bg2T+jD8qmNSVPBKUd/wDgmlXB0cZxPOjX+G97d7RTsVNMh+Jvj+I3sGpz2tnIxUSeebeIjoQFTkgfQ/iareKvh94x0Pwzd6jrOuxXdlDs8yFbyZy2XVR8rKAeSD+Fes+AfEujah4L01LW7t4pLa2jhmgZwrRsqgHIPbIznvXI/Fvx7pVz4fuPDmkyrf3NwVMzwtuSFUYP1HBPy9B0Gc1E6NJUeeUm2136nThcxx08zWFo0IwhGWqUdop6tv06/cHwC/5Aur/9fCf+gmvMv2sBL/wmehFv9V/Z7bef4vMOf0216b8Av+QLq/8A18J/6Cayf2n/AAdc634NsvEFhG0smiu/2hFGT5MmMt/wEqv4EntXbg/4ET5niX/kbVvVfkjhPDvwa+KureF9L1DSfGkFvY3VnFNbQnVbpfLjZAVXaqEDAIGBwMVpf8KK+MX/AEPdv/4OLz/43W98B/jJoR8HWnhjxPqMOnX2njyreW6cJHPFn5RvPAYZ24PYDGecey3Pi3w5Z2/n3ev6XBD/AM9JLyNV/Mmus+fPDvh1+z94s8NfEqw8S+IdZ02eO2leeU280ssszsrDkui9S2SSa9h+JP8AyTfWv+vf+oqLw/8AE/wl4q8TTaD4e1VL+8hgM7mJG8vaGCkByMMfmHTP6VL8Sf8Akm+tf9e/9RWVb+HL0Z3Zd/vtH/FH80cB+z7/AMzD/wBu3/tWvZq+Z/hv42fwTq0kt1A0mmXxEc5VfmUr0ZT3I3nI9D9K9/i8YeHZ9O+3R63Ym227i5nUY+oPIPtjNceBqw9io31R9JxVgMSsylXUG4ztZrXZJW9dDyz4/tH9u0NRnzRHMW/3cpj+TV3HhFXHwatAA246bJgY56NivHfGuty/Eb4gRRaNG8kR22tmpGCwySWI7ZJJ9gK+i9N06LTNGtdNiAaG2gSAZH3gqgcj3xU4f97XqVI7bG+cXwWU4TC1PjT5mu27/W3yPA/geyD4gsHHLWUgXnGDlfz4zX0PXzMftfwv+KG94WaO1mJVf+e0DZHB9dp/Aj2r37S/GPh7WLJbqy1e0KMMlZJVR09mUnINGAmowdOWjTFxZh6lfEQxtJOVOcVZrX+tCTxa0aeC9aab/V/YJ92e48tq8c+AqufF2osD8gsSCMdzImOfwNb/AMWPiLpj6DNoOiXUd5cXWFnlhbckSA5I3Dgk4xgdBmr/AMFPDE2j+HJ9VvYzHPqRUxqw5ES5wfxJJ+gFE5Kri4qP2dww9OeAyCtKurOq0op7vz/P7ih8ff8AkC6R/wBfD/8AoIrqPhH/AMks0j/tt/6PeuX+Pv8AyBdI/wCvh/8A0EV1Hwj/AOSWaR/22/8AR71UP99l6f5GOK/5Jih/18f/ALeeQftc/wDMo/8Ab7/7QrzPxd8O7/SfAPhrxvYvLdWF5aRC58z959llHC5z/AcADsDx3Ar0z9rn/mUf+33/ANoV6n8MNMs9Z+A+habqcCXFpdaaIponHDKcg/8A669E+NI/g78Tbb4jeE1aXy4dYsVWO9t14HTiRR/dbHTscj0J9Dr4x8QaRr37PvxYgvtLd5bJmL2sr8LdW5I3RP7jgH32sMcV9Z+EPFem+NfDFprmjSbre4XlG+9E4+8jehB/x70AeEfDFgnxhsvOyG824A3HGG8t/wD9VfSVfNvjCxvPAXxSOoW8WIzc/bbU/wALqWyUz7ZKn/69e46F468PeINPS5tdSt4nKgvBPKqSRn0IJ/UcV5eBkqfNSlo0z7riihUxXscfRXNCUEtNbO7ev3/gbly0aWkzTjMSoxcf7OOa+dvgurN8SLcr0W3lLcZ424/DkivRfiV8SNK0/wAPXWmaRexXmoXcZh/cOHWFWGGJYcZxnA61k/AzwxNbQXfiG7jKC4TyLbcPvJnLN9CQAPoadZqriYRj01ZOXU54DJcVWrrl9paMU+u+v4/gev0UUV6Z8OFch4t+GujeMtVh1DVJ72KaKEQgW8iqpUMW5yp7sa6+ionCM1yyV0dGGxVbC1PaUJOMu6I7eBLW1it4siOJAi5OeAMCotRsYtT0u6sLnd5N1C8Mm04O1lKnB9cGrNFVZWsYqclLnT13OW8IfD/SvBUt1JpU13M10qq5uXVsAZxjao9a1/EOhWviXQbnSNQaVLe5C72hYBhtYMMEgjqB2rSoqFThGPIlodFTGYirX+szm3PR366bfcc94R8F6b4Ls7i20qS5lW4cO7XDhjkDA6ACr2v+HtO8TaU+n6vB5sLHcCDhkbsynsa06KapxUeRLQU8XXnX+sSm+fe/W54/P8ALZrjNr4gmjhz9yS1Dtj/eDD+VdXo3wr8PaPot3Yqss8t5C0M125HmBT1C8YX8vrmu1orGOFowd1E9Kvn2Z4iChUrOy9F+SV/mc94R8F6b4Ls7i30qW5lW4cO7XDqxyBgYwAK6B0WSNkkUOjAhlYZBHoaWit4xjBcsdjyq9epiKjq1XeT3Z4v4q/Zk8J65eSXeiXVzoUshJMUKiSAH2Q4I+gbHoK5u2/ZItkmzeeMZZY/7sWnCM/mZG/lX0ZRVGJwfw++D/hj4cXEt5owuri/mi8l7u6l3NsJBKgKAoGVHbPHWuu1rSbfXdFutMvTIsFymxzGcMB7Eg1eopNJqzLhOVOSnF2a1Rx2n/C7w5Y+HLjRZIp7y2uJfOL3DgyI+MAqVAxx/9euQuPgBatclrXxBLHBnhJLUOwH+8GA/SvYKK55YWjJJOOx69HPsyoSlKnWd5b3s/wA0zk/B3w60bwaWmsw9zeuu1rqbG4D0UDhR+vvXWUUVtCEYLlirI83EYmtiqjq15OUn1ZieJvCGj+LbNYNYty7R58qaM7ZI8+h/ocivObj9n+BpSbXxFJHH2WS0Dn8w4/lXsNFZVMPSqu84ndg85zDAw5MPVaXbRr8U7HnXh74L6Bo90tzqEsuqyocqsyhYx9VHX8Tj2r0UDAwOBRRWlOlCmrQVjlxeOxONnz4ibk/62WyOe8XeC9N8aWdvb6rLcxLbuXRrd1U5IxzlTV/w/oVr4a0G20nTzI1vbBghlYFjlixJIA7k9q0qKapxUue2pEsVXlQWHcnyJ3S6X/pnFfEX4V6F8TY9PXX7i/gOnmQwtZSIhO/buB3KwP3Fro/DuhWnhjw7Y6LppkNrYwiGIytuYgdyeOfwrSoqzmOe8a+CNF8feH20jxBC7w7xJHLEwWSFx/EpIODgkcgjms34efC/R/hpBfQ6Fe6lcR3rK0i3syuFKgjKhVUAnPJxk4HpXZ0UAZfiDw3pfifTTZazbCeMHcjA4aNvVSOhrzS7+AFo8xNjr80MeeFmthIfzDL/ACr1+isKmHpVXeaPVweb47Ax5cPUaXbRr7nc8y0P4HaHp1ys+q3c2plCCIyojjJ9wCSfpmvS440hiWOJFSNFCqijAUDoAOwp1FVTo06StBWMMZmGKx0lLEzcrfd9y0CiiitThP/Z)**

**Input:** text = "nlaebolko"

**Output:** 1

class Solution:

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

return int(min(text.count("b"),text.count("a"),text.count("l")//2,text.count("o")//2,text.count("n")))

**1196. How Many Apples Can You Put into the Basket**

You have some apples and a basket that can carry up to 5000 units of weight.

Given an integer array weight where weight[i] is the weight of the ith apple, return *the maximum number of apples you can put in the basket*.

**Example 1:**

**Input:** weight = [100,200,150,1000]

**Output:** 4

**Explanation:** All 4 apples can be carried by the basket since their sum of weights is 1450.

**My sol:**

class Solution:

def maxNumberOfApples(self, weight: List[int]) -> int:

count=0

sum=0

for x in sorted(weight):

if sum+x<=5000:

count=count+1

sum+=x

return count

**1198. Find Smallest Common Element in All Rows**

Given an m x n matrix mat where every row is sorted in **strictly** **increasing** order, return *the****smallest common element****in all rows*.

If there is no common element, return -1.

**Example 1:**

**Input:** mat = [[1,2,3,4,5],[2,4,5,8,10],[3,5,7,9,11],[1,3,5,7,9]]

**Output:** 5

**My sol:**

class Solution:

def smallestCommonElement(self, mat: List[List[int]]) -> int:

flag=0

a=[]

for y in mat[0]:

for x in range(1,len(mat)):

if y in mat[x]:

flag=0

else:

flag=1

break

if flag==0 and y not in a:

a.append(y)

if a:

return min(a)

return -1

**1200. Minimum Absolute Difference**

Given an array of **distinct** integers arr, find all pairs of elements with the minimum absolute difference of any two elements.

Return a list of pairs in ascending order(with respect to pairs), each pair [a, b] follows

* a, b are from arr
* a < b
* b - a equals to the minimum absolute difference of any two elements in arr

**Example 1:**

**Input:** arr = [4,2,1,3]

**Output:** [[1,2],[2,3],[3,4]]

**Explanation:** The minimum absolute difference is 1. List all pairs with difference equal to 1 in ascending order.

**My sol:**

class Solution:

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

minval=10000

temp=[]

a=sorted(arr)

aa=collections.Counter(arr)

for x in range(0,len(a)-1):

minval=min(minval,abs(a[x+1]-a[x]))

for x in a:

if x+minval in aa:

temp.append([x,x+minval])

return temp

1201-1250 ($$$$$ 6)

**1207. Unique Number of Occurrences**

Given an array of integers arr, return true if the number of occurrences of each value in the array is **unique**, or false otherwise.

**Example 1:**

**Input:** arr = [1,2,2,1,1,3]

**Output:** true

**Explanation:** The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.

**My sol:**

class Solution:

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

flag=0

dict={}

result=[]

for x in arr:

if x in dict:

dict[x]+=1

else:

dict[x]=1

for key,value in dict.items():

result.append(value)

print(result)

for y in range(0,len(result)):

if result[y] in result[y+1:]:

flag=1

break

return bool(flag==0)

**1213. Intersection of Three Sorted Arrays**

Given three integer arrays arr1, arr2 and arr3 **sorted** in **strictly increasing** order, return a sorted array of **only** the integers that appeared in **all** three arrays.

**Example 1:**

**Input:** arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]

**Output:** [1,5]

**Explanation:** Only 1 and 5 appeared in the three arrays.

**My sol:**

class Solution:

def arraysIntersection(self, arr1: List[int], arr2: List[int], arr3: List[int]) -> List[int]:

s=[]

for x in arr1:

if x in arr2 and x in arr3:

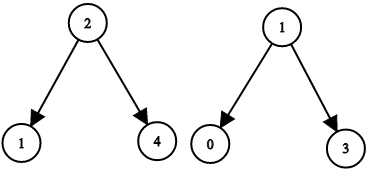
s.append(x)

return s

**1214. Two Sum BSTs**

Given the roots of two binary search trees, root1 and root2, return true if and only if there is a node in the first tree and a node in the second tree whose values sum up to a given integer target.

**Example 1:**



**Input:** root1 = [2,1,4], root2 = [1,0,3], target = 5

**Output:** true

**Explanation:** 2 and 3 sum up to 5.

**My sol:**

class Solution:

def twoSumBSTs(self, root1: Optional[TreeNode], root2: Optional[TreeNode], target: int) -> bool:

a=[]

b=[]

def temp1(root):

a.append(root.val)

if root.left:

temp1(root.left)

if root.right:

temp1(root.right)

temp1(root1)

def temp2(root):

b.append(root.val)

if root.left:

temp2(root.left)

if root.right:

temp2(root.right)

temp2(root2)

print(a)

print(b)

# if len(a)==1:

# return False

for x in a:

if target-x in b and target-x !=x:

return True

return False

**1221. Split a String in Balanced Strings**

**Balanced** strings are those that have an equal quantity of 'L' and 'R' characters.

Given a **balanced** string s, split it into some number of substrings such that:

* Each substring is balanced.

Return *the****maximum****number of balanced strings you can obtain.*

**Example 1:**

**Input:** s = "RLRRLLRLRL"

**Output:** 4

**Explanation:** s can be split into "RL", "RRLL", "RL", "RL", each substring contains same number of 'L' and 'R'.

My sol:

class Solution:

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

# If val = s[0] we increment counter by 1, else we decrement counter by 1

# Counter variable = 0 means we have found one split so we increment the "res" variable

val = s[0]

counter = 1 # To check for a valid split

res = 0 # To count the number of splits

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

if s[i] == val:

counter += 1

else:

counter -= 1

if counter == 0:

res += 1

return res

**1228. Missing Number In Arithmetic Progression**

In some array arr, the values were in arithmetic progression: the values arr[i + 1] - arr[i] are all equal for every 0 <= i < arr.length - 1.

A value from arr was removed that **was not the first or last value in the array**.

Given arr, return *the removed value*.

**Example 1:**

**Input:** arr = [5,7,11,13]

**Output:** 9

**Explanation:** The previous array was [5,7,**9**,11,13].

**My sol:**

class Solution:

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

b=sorted(arr)

ma=max(arr)

mi=min(arr)

s=((len(arr)+1)\*(ma+mi))/2

print(s)

return int(s-sum(arr))

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

You are given an array coordinates, coordinates[i] = [x, y], where [x, y] represents the coordinate of a point. Check if these points make a straight line in the XY plane.

**Example 1:**

Chart

Description automatically generated

**Input:** coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

**Output:** true

My sol:

class Solution:

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

# print(coordinates[0])

x0, y0 = coordinates[0]

x1, y1 = coordinates[1]

dy = y1 - y0

dx = x1 - x0

xp = dx\*y0 - dy\*x0

print(xp)

for x, y in coordinates:

if dx\*y - dy\*x != xp:

return False

else:

return True

1251-1300 ($$$$$ 9)

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

There is an m x n matrix that is initialized to all 0's. There is also a 2D array indices where each indices[i] = [ri, ci] represents a **0-indexed location** to perform some increment operations on the matrix.

For each location indices[i], do **both** of the following:

1. Increment **all** the cells on row ri.
2. Increment **all** the cells on column ci.

Given m, n, and indices, return *the****number of odd-valued cells****in the matrix after applying the increment to all locations in*indices.

**Example 1:**

A picture containing text, clock, gauge

Description automatically generated

**Input:** m = 2, n = 3, indices = [[0,1],[1,1]]

**Output:** 6

**Explanation:** Initial matrix = [[0,0,0],[0,0,0]].

After applying first increment it becomes [[1,2,1],[0,1,0]].

The final matrix is [[1,3,1],[1,3,1]], which contains 6 odd numbers.

**My sol:**

class Solution:

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

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

# print(a)

r=[]

c=[]

count=0

for x in indices:

r.append(x[0])

c.append(x[1])

for x in range(0,len(a)):

for y in range(0,len(a[x])):

if x in r:

for temp in range(r.count(x)):

a[x][y]+=1

if y in c:

for t in range(c.count(y)):

a[x][y]+=1

for x in range(0,len(a)):

for y in range(0,len(a[x])):

if a[x][y]%2!=0:

count+=1

return count

**1265. Print Immutable Linked List in Reverse**

You are given an immutable linked list, print out all values of each node in reverse with the help of the following interface:

* ImmutableListNode: An interface of immutable linked list, you are given the head of the list.

You need to use the following functions to access the linked list (you **can't** access the ImmutableListNode directly):

* ImmutableListNode.printValue(): Print value of the current node.
* ImmutableListNode.getNext(): Return the next node.

The input is only given to initialize the linked list internally. You must solve this problem without modifying the linked list. In other words, you must operate the linked list using only the mentioned APIs.

**Example 1:**

**Input:** head = [1,2,3,4]

**Output:** [4,3,2,1]

**My sol:**

class Solution:

def printLinkedListInReverse(self, head: 'ImmutableListNode') -> None:

if head is not None:

self.printLinkedListInReverse(head.getNext())

head.printValue()

**1271. Hexspeak**

A decimal number can be converted to its **Hexspeak representation** by first converting it to an uppercase hexadecimal string, then replacing all occurrences of the digit '0' with the letter 'O', and the digit '1' with the letter 'I'. Such a representation is valid if and only if it consists only of the letters in the set {'A', 'B', 'C', 'D', 'E', 'F', 'I', 'O'}.

Given a string num representing a decimal integer n, *return the****Hexspeak representation****of*n*if it is valid, otherwise return*"ERROR".

**Example 1:**

**Input:** num = "257"

**Output:** "IOI"

**Explanation:** 257 is 101 in hexadecimal.

**My sol:**

class Solution:

def toHexspeak(self, num: str) -> str:

flag=0

a=hex(int(num))[2:]

print(a)

b=""

for x in a:

if x=='1':

b+="I"

elif x=='0':

b+="O"

elif x=='a':

b+='A'

elif x=='b':

b+='B'

elif x=='c':

b+='C'

elif x=='d':

b+='D'

elif x=='e':

b+='E'

elif x=='f':

b+='F'

else:

flag=1

break

if flag==1:

return "ERROR"

if b=="":

return "ERROR"

return b

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

Given an integer number n, return the difference between the product of its digits and the sum of its digits.

**Example 1:**

**Input:** n = 234

**Output:** 15

**Explanation:**

Product of digits = 2 \* 3 \* 4 = 24

Sum of digits = 2 + 3 + 4 = 9

Result = 24 - 9 = 15

**My sol:**

class Solution:

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

product =1

sum=0

for j in str(n):

product\*=int(j)

sum+=int(j)

result = product-sum

return result

**1282. Group the People Given the Group Size They Belong To**

There are n people that are split into some unknown number of groups. Each person is labeled with a **unique ID** from 0 to n - 1.

You are given an integer array groupSizes, where groupSizes[i] is the size of the group that person i is in. For example, if groupSizes[1] = 3, then person 1 must be in a group of size 3.

Return *a list of groups such that each person i is in a group of size groupSizes[i]*.

Each person should appear in **exactly one group**, and every person must be in a group. If there are multiple answers, **return any of them**. It is **guaranteed** that there will be **at least one** valid solution for the given input.

**Example 1:**

**Input:** groupSizes = [3,3,3,3,3,1,3]

**Output:** [[5],[0,1,2],[3,4,6]]

**Explanation:**

The first group is [5]. The size is 1, and groupSizes[5] = 1.

The second group is [0,1,2]. The size is 3, and groupSizes[0] = groupSizes[1] = groupSizes[2] = 3.

The third group is [3,4,6]. The size is 3, and groupSizes[3] = groupSizes[4] = groupSizes[6] = 3.

Other possible solutions are [[2,1,6],[5],[0,4,3]] and [[5],[0,6,2],[4,3,1]].

**My sol:**

class Solution:

def groupThePeople(self, groupSizes: List[int]) -> List[List[int]]:

dict={}

for x in range(0,len(groupSizes)):

if groupSizes[x] in dict:

dict[groupSizes[x]].append(x)

else:

dict[groupSizes[x]]=[x]

print(dict)

res=[]

temp=[]

for k,v in dict.items():

for y in v:

if len(temp)==k:

res.append(temp)

temp=[]

temp.append(y)

if temp!=[]:

res.append(temp)

temp=[]

return res

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

Given an integer array **sorted** in non-decreasing order, there is exactly one integer in the array that occurs more than 25% of the time, return that integer.

**Example 1:**

**Input:** arr = [1,2,2,6,6,6,6,7,10]

**Output:** 6

**My sol:**

class Solution:

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

dic={}

for x in arr:

if x in dic:

dic[x]=dic[x]+1

else:

dic[x]=1

maxvalue=max(dic,key=dic.get)

return maxvalue

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

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the *decimal value* of the number in the linked list.

The **most significant bit** is at the head of the linked list.

**Example 1:**

A picture containing text, clock

Description automatically generated

**Input:** head = [1,0,1]

**Output:** 5

**Explanation:** (101) in base 2 = (5) in base 10

My sol:

class Solution:

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

a=""

while(head):

a=a+str(head.val)

# print(head.val)

head=head.next

return (int(a,2))

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

Given an array nums of integers, return how many of them contain an **even number** of digits.

**Example 1:**

**Input:** nums = [12,345,2,6,7896]

**Output:** 2

**Explanation:**

12 contains 2 digits (even number of digits).

345 contains 3 digits (odd number of digits).

2 contains 1 digit (odd number of digits).

6 contains 1 digit (odd number of digits).

7896 contains 4 digits (even number of digits).

Therefore only 12 and 7896 contain an even number of digits.

**My sol:**

class Solution:

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

#rev=0

even=0

a=[]

for i in nums:

count=0

while(i>0):

#rem=i%10

#rev=rev\*10 +rem

count=count+1

i=i//10

a.append(count)

#print(count)

#print(a)

for j in a:

if j%2==0:

even=even+1

return even

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

Given an array arr, replace every element in that array with the greatest element among the elements to its right, and replace the last element with -1.

After doing so, return the array.

**Example 1:**

**Input:** arr = [17,18,5,4,6,1]

**Output:** [18,6,6,6,1,-1]

**Explanation:**

- index 0 --> the greatest element to the right of index 0 is index 1 (18).

- index 1 --> the greatest element to the right of index 1 is index 4 (6).

- index 2 --> the greatest element to the right of index 2 is index 4 (6).

- index 3 --> the greatest element to the right of index 3 is index 4 (6).

- index 4 --> the greatest element to the right of index 4 is index 5 (1).

- index 5 --> there are no elements to the right of index 5, so we put -1.

**My sol:**

class Solution:

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

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

arr[i]= max(arr[i+1 :])

arr[len(arr)-1]=-1

return arr

1301-1350 ($$$$$ 18)

**1302. Deepest Leaves Sum**

Given the root of a binary tree, return *the sum of values of its deepest leaves*.

**Example 1:**

Diagram

Description automatically generated

**Input:** root = [1,2,3,4,5,null,6,7,null,null,null,null,8]

**Output:** 15

**My sol:**

class Solution:

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

res=[]

q=collections.deque()

q.append(root)

while q:

qlen=len(q)

level=[]

for x in range(qlen):

node=q.popleft()

if node:

level.append(node.val)

q.append(node.left)

q.append(node.right)

if level:

res.append(level)

return sum(res[-1])

**1304. Find N Unique Integers Sum up to Zero**

Given an integer n, return **any** array containing n **unique** integers such that they add up to 0.

**Example 1:**

**Input:** n = 5

**Output:** [-7,-1,1,3,4]

**Explanation:** These arrays also are accepted [-5,-1,1,2,3] , [-3,-1,2,-2,4].

**My sol:**

class Solution:

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

return range(1 - n, n, 2)

**1305. All Elements in Two Binary Search Trees**

Given two binary search trees root1 and root2, return *a list containing all the integers from both trees sorted in****ascending****order*.

**Example 1:**

Diagram, shape

Description automatically generated

**Input:** root1 = [2,1,4], root2 = [1,0,3]

**Output:** [0,1,1,2,3,4]

**My sol:**

class Solution:

def getAllElements(self, root1: TreeNode, root2: TreeNode) -> List[int]:

a=[]

def temp(root):

if root:

a.append(root.val)

if root.left:

temp(root.left)

if root.right:

temp(root.right)

if not root1:

pass

else:

temp(root1)

if not root2:

pass

else:

temp(root2)

return sorted(a)

**1309. Decrypt String from Alphabet to Integer Mapping**

You are given a string s formed by digits and '#'. We want to map s to English lowercase characters as follows:

* Characters ('a' to 'i') are represented by ('1' to '9') respectively.
* Characters ('j' to 'z') are represented by ('10#' to '26#') respectively.

Return *the string formed after mapping*.

The test cases are generated so that a unique mapping will always exist.

**Example 1:**

**Input:** s = "10#11#12"

**Output:** "jkab"

**Explanation:** "j" -> "10#" , "k" -> "11#" , "a" -> "1" , "b" -> "2".

**My sol:**

class Solution:

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

output =""

if '26#' in s:

s=s.replace('26#','z')

if '25#' in s:

s=s.replace('25#','y')

if '24#' in s:

s=s.replace('24#','x')

if '23#' in s:

s=s.replace('23#','w')

if '22#' in s:

s=s.replace('22#','v')

if '21#' in s:

s=s.replace('21#','u')

if '20#' in s:

s=s.replace('20#','t')

if '19#' in s:

s=s.replace('19#','s')

if '18#' in s:

s=s.replace('18#','r')

if '17#' in s:

s=s.replace('17#','q')

if '16#' in s:

s=s.replace('16#','p')

if '15#' in s:

s=s.replace('15#','o')

if '14#' in s:

s=s.replace('14#','n')

if '13#' in s:

s=s.replace('13#','m')

if '12#' in s:

s=s.replace('12#','l')

if '11#' in s:

s=s.replace('11#','k')

if '10#' in s:

s=s.replace('10#','j')

if '9' in s:

s=s.replace('9','i')

if '8' in s:

s=s.replace('8','h')

if '7' in s:

s=s.replace('7','g')

if '6' in s:

s=s.replace('6','f')

if '5' in s:

s=s.replace('5','e')

if '4' in s:

s=s.replace('4','d')

if '3' in s:

s=s.replace('3','c')

if '2' in s:

s=s.replace('2','b')

if '1' in s:

s=s.replace('1','a')

return s

**1313. Decompress Run-Length Encoded List**

We are given a list nums of integers representing a list compressed with run-length encoding.

Consider each adjacent pair of elements [freq, val] = [nums[2\*i], nums[2\*i+1]] (with i >= 0).  For each such pair, there are freq elements with value val concatenated in a sublist. Concatenate all the sublists from left to right to generate the decompressed list.

Return the decompressed list.

**Example 1:**

**Input:** nums = [1,2,3,4]

**Output:** [2,4,4,4]

**Explanation:** The first pair [1,2] means we have freq = 1 and val = 2 so we generate the array [2].

The second pair [3,4] means we have freq = 3 and val = 4 so we generate [4,4,4].

At the end the concatenation [2] + [4,4,4] is [2,4,4,4].

**My sol:**

class Solution:

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

result=[]

for i in range(0,len(nums),2):

result=result+[nums[i+1]]\*nums[i]

return(result)

**1317. Convert Integer to the Sum of Two No-Zero Integers**

**No-Zero integer** is a positive integer that **does not contain any 0** in its decimal representation.

Given an integer n, return *a list of two integers* [A, B] *where*:

* A and B are **No-Zero integers**.
* A + B = n

The test cases are generated so that there is at least one valid solution. If there are many valid solutions you can return any of them.

**Example 1:**

**Input:** n = 2

**Output:** [1,1]

**Explanation:** A = 1, B = 1. A + B = n and both A and B do not contain any 0 in their decimal representation.

**My sol:**

class Solution:

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

a=1

while '0' in str(n-a) or '0' in str(a):

a+=1

return [a,n-a]

**1324. Print Words Vertically**

Given a string s. Return all the words vertically in the same order in which they appear in s.  
Words are returned as a list of strings, complete with spaces when is necessary. (Trailing spaces are not allowed).  
Each word would be put on only one column and that in one column there will be only one word.

**Example 1:**

**Input:** s = "HOW ARE YOU"

**Output:** ["HAY","ORO","WEU"]

**Explanation:** Each word is printed vertically.

"HAY"

 "ORO"

 "WEU"

**My sol:**

class Solution:

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

a=s.split()

final=[]

temp=[]

i=0

str=""

m=0

for x in a:

if len(x)>m:

m=len(x)

while(i<m):

for x in a:

# print(x[i])

if i>len(x)-1:

str+=" "

else:

# print(i)

str+=x[i]

i+=1

t=len(str)

while str[t-1]==" ":

t-=1

final.append(str[:t])

str=""

return final

**1323. Maximum 69 Number**

You are given a positive integer num consisting only of digits 6 and 9.

Return *the maximum number you can get by changing****at most****one digit (*6*becomes*9*, and*9*becomes*6*)*.

**Example 1:**

**Input:** num = 9669

**Output:** 9969

**Explanation:**

Changing the first digit results in 6669.

Changing the second digit results in 9969.

Changing the third digit results in 9699.

Changing the fourth digit results in 9666.

The maximum number is 9969.

**My sol:**

class Solution:

def maximum69Number (self, num: int) -> int:

for i in str(num):

if int(i)==9:

pass

else:

num=str(num).replace(i,str(9), 1)

break

return num

**1329. Sort the Matrix Diagonally**

A **matrix diagonal** is a diagonal line of cells starting from some cell in either the topmost row or leftmost column and going in the bottom-right direction until reaching the matrix's end. For example, the **matrix diagonal** starting from mat[2][0], where mat is a 6 x 3 matrix, includes cells mat[2][0], mat[3][1], and mat[4][2].

Given an m x n matrix mat of integers, sort each **matrix diagonal** in ascending order and return *the resulting matrix*.

**Example 1:**

Shape, line chart

Description automatically generated

**Input:** mat = [[3,3,1,1],[2,2,1,2],[1,1,1,2]]

**Output:** [[1,1,1,1],[1,2,2,2],[1,2,3,3]]

**My sol:**

class Solution:

def diagonalSort(self, mat: List[List[int]]) -> List[List[int]]:

d = {}

rows = len(mat)

cols = len(mat[0])

for i in range(rows):

for j in range(cols):

if i-j in d:

d[i-j].append(mat[i][j])

else:

d[i-j] = [mat[i][j]]

print(d)

for k in d.keys():

d[k].sort(reverse=True)

print(d)

for i in range(rows):

for j in range(cols):

mat[i][j] = d[i-j].pop()

return mat

**1331. Rank Transform of an Array**

Given an array of integers arr, replace each element with its rank.

The rank represents how large the element is. The rank has the following rules:

* Rank is an integer starting from 1.
* The larger the element, the larger the rank. If two elements are equal, their rank must be the same.
* Rank should be as small as possible.

**Example 1:**

**Input:** arr = [40,10,20,30]

**Output:** [4,1,2,3]

**Explanation**: 40 is the largest element. 10 is the smallest. 20 is the second smallest. 30 is the third smallest.

**My sol:**

class Solution:

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

s=sorted(list(set(arr)))

dict={}

c=0

for x in range(0,len(s)):

dict[s[x]]=x+1

print(dict)

for x in range(0,len(arr)):

arr[x]=dict[arr[x]]

return arr

**1332. Remove Palindromic Subsequences**

You are given a string s consisting **only** of letters 'a' and 'b'. In a single step you can remove one **palindromic subsequence** from s.

Return *the****minimum****number of steps to make the given string empty*.

A string is a **subsequence** of a given string if it is generated by deleting some characters of a given string without changing its order. Note that a subsequence does **not** necessarily need to be contiguous.

A string is called **palindrome** if is one that reads the same backward as well as forward.

**Example 1:**

**Input:** s = "ababa"

**Output:** 1

**Explanation:** s is already a palindrome, so its entirety can be removed in a single step.

**My sol:**

class Solution:

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

if s==s[::-1]:

return 1

if s=="":

return 0

else:

return 2

**1337. The K Weakest Rows in a Matrix**

You are given an m x n binary matrix mat of 1's (representing soldiers) and 0's (representing civilians). The soldiers are positioned **in front** of the civilians. That is, all the 1's will appear to the **left** of all the 0's in each row.

A row i is **weaker** than a row j if one of the following is true:

* The number of soldiers in row i is less than the number of soldiers in row j.
* Both rows have the same number of soldiers and i < j.

Return *the indices of the*k***weakest****rows in the matrix ordered from weakest to strongest*.

**Example 1:**

**Input:** mat =

[[1,1,0,0,0],

[1,1,1,1,0],

[1,0,0,0,0],

[1,1,0,0,0],

[1,1,1,1,1]],

k = 3

**Output:** [2,0,3]

**Explanation:**

The number of soldiers in each row is:

- Row 0: 2

- Row 1: 4

- Row 2: 1

- Row 3: 2

- Row 4: 5

The rows ordered from weakest to strongest are [2,0,3,1,4].

**My sol:**

class Solution:

def kWeakestRows(self, mat: List[List[int]], k: int) -> List[int]:

dict={}

for x in range(0,len(mat)):

dict[x]=mat[x].count(1)

# print(dict)

temp={k: v for k, v in sorted(dict.items(), key=lambda item: item[1])}

print(temp)

fin=[]

for key,value in temp.items():

fin.append(key)

return fin[:k]

**1338. Reduce Array Size to The Half**

You are given an integer array arr. You can choose a set of integers and remove all the occurrences of these integers in the array.

Return *the minimum size of the set so that****at least****half of the integers of the array are removed*.

**Example 1:**

**Input:** arr = [3,3,3,3,5,5,5,2,2,7]

**Output:** 2

**Explanation:** Choosing {3,7} will make the new array [5,5,5,2,2] which has size 5 (i.e equal to half of the size of the old array).

Possible sets of size 2 are {3,5},{3,2},{5,2}.

Choosing set {2,7} is not possible as it will make the new array [3,3,3,3,5,5,5] which has a size greater than half of the size of the old array.

**My sol:**

class Solution:

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

d={}

for x in arr:

if x in d:

d[x]+=1

else:

d[x]=1

a={k: v for k, v in sorted(d.items(), key=lambda item: -item[1])}

l=len(arr)

if len(a)==1:

return 1

count=0

temp=[]

for k,v in a.items():

if count<l/2:

count+=v

temp.append(k)

return len(temp)

**1342. Number of Steps to Reduce a Number to Zero**

Given an integer num, return *the number of steps to reduce it to zero*.

In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

**Example 1:**

**Input:** num = 14

**Output:** 6

**Explanation:**

Step 1) 14 is even; divide by 2 and obtain 7.

Step 2) 7 is odd; subtract 1 and obtain 6.

Step 3) 6 is even; divide by 2 and obtain 3.

Step 4) 3 is odd; subtract 1 and obtain 2.

Step 5) 2 is even; divide by 2 and obtain 1.

Step 6) 1 is odd; subtract 1 and obtain 0.

**My sol:**

class Solution:

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

count=0

while num!=0 :

if num!=0 and num%2==0:

num=num/2

count+=1

elif num!=0 and num%2==1:

num=(num-1)

count+=1

elif num==0:

pass

return count

**1343. Number of Sub-arrays of Size K and Average Greater than or Equal to Threshold**

**(sliding window)**

Given an array of integers arr and two integers k and threshold, return *the number of sub-arrays of size*k*and average greater than or equal to*threshold.

**Example 1:**

**Input:** arr = [2,2,2,2,5,5,5,8], k = 3, threshold = 4

**Output:** 3

**Explanation:** Sub-arrays [2,5,5],[5,5,5] and [5,5,8] have averages 4, 5 and 6 respectively. All other sub-arrays of size 3 have averages less than 4 (the threshold).

**My sol:**

class Solution:

def numOfSubarrays(self, arr: List[int], k: int, threshold: int) -> int:

i=0

count=0

temp=sum(arr[i:i+k-1])

while i+k<=len(arr):

temp+=arr[i+k-1]

if temp>=k\*threshold:

count+=1

temp-=arr[i]

i+=1

return count

**1344. Angle Between Hands of a Clock**

Given two numbers, hour and minutes, return *the smaller angle (in degrees) formed between the*hour*and the*minute*hand*.

Answers within 10-5 of the actual value will be accepted as correct.

**Example 1:**

A black and white clock

Description automatically generated with medium confidence

**Input:** hour = 12, minutes = 30

**Output:** 165

**My sol:**

class Solution:

def angleClock(self, hour: int, minutes: int) -> float:

a=(hour\*30+((minutes/60)\*30))%360

b=minutes\*6

return min(abs(b-a),360-abs(b-a))

**1346. Check If N and Its Double Exist**

Given an array arr of integers, check if there exists two integers N and M such that N is the double of M ( i.e. N = 2 \* M).

More formally check if there exists two indices i and j such that :

* i != j
* 0 <= i, j < arr.length
* arr[i] == 2 \* arr[j]

**Example 1:**

**Input:** arr = [10,2,5,3]

**Output:** true

**Explanation:** N = 10 is the double of M = 5,that is, 10 = 2 \* 5.

**My sol:**

class Solution:

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

cpy =arr;

flag=0;

for i in range(0,len(cpy)):

for j in range(0,len(cpy)):

# print(2\*cpy[i] , cpy[j]);

if (2\*cpy[i] == cpy[j] and i!=j) :

print(2\*cpy[i] , cpy[j],i,j)

flag=1;

return flag==1;

else:

flag=0;

if flag==0:

return flag==1

**1347. Minimum Number of Steps to Make Two Strings Anagram**

You are given two strings of the same length s and t. In one step you can choose **any character** of t and replace it with **another character**.

Return *the minimum number of steps* to make t an anagram of s.

An **Anagram** of a string is a string that contains the same characters with a different (or the same) ordering.

**Example 1:**

**Input:** s = "bab", t = "aba"

**Output:** 1

**Explanation:** Replace the first 'a' in t with b, t = "bba" which is anagram of s.

**My sol:**

class Solution:

def minSteps(self, s: str, t: str) -> int:

dict1=collections.Counter(s)

dict2=collections.Counter(t)

count=0

for x in dict2.keys():

if x in dict1:

if dict2[x]-dict1[x]<0:

pass

else:

count+=abs(dict2[x]-dict1[x])

else:

count+=dict2[x]

return count

1351-1400 ($$$$$ 15)

**1351. Count Negative Numbers in a Sorted Matrix**

Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return *the number of****negative****numbers in* grid.

**Example 1:**

**Input:** grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

**Output:** 8

**Explanation:** There are 8 negatives number in the matrix.

**My sol:**

class Solution:

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

count=0

for i in range(0,len(grid)):

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

if grid[i][j]>=0:

pass

else:

count=count+1

return count

**1356. Sort Integers by The Number of 1 Bits**

You are given an integer array arr. Sort the integers in the array in ascending order by the number of 1's in their binary representation and in case of two or more integers have the same number of 1's you have to sort them in ascending order.

Return *the array after sorting it*.

**Example 1:**

**Input:** arr = [0,1,2,3,4,5,6,7,8]

**Output:** [0,1,2,4,8,3,5,6,7]

**Explantion:** [0] is the only integer with 0 bits.

[1,2,4,8] all have 1 bit.

[3,5,6] have 2 bits.

[7] has 3 bits.

The sorted array by bits is [0,1,2,4,8,3,5,6,7]

**My sol:**

class Solution:

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

return sorted(arr, key=lambda x: (bin(x).count("1"), x))

**1358. Number of Substrings Containing All Three Characters**

Given a string s consisting only of characters *a*, *b* and *c*.

Return the number of substrings containing **at least** one occurrence of all these characters *a*, *b* and *c*.

**Example 1:**

**Input:** s = "abcabc"

**Output:** 10

**Explanation:** The substrings containing at least one occurrence of the characters *a*, *b* and *c are "*abc*", "*abca*", "*abcab*", "*abcabc*", "*bca*", "*bcab*", "*bcabc*", "*cab*", "*cabc*"* and *"*abc*"* (**again**)*.*

**My sol:**

class Solution:

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

d={}

res=0

left=0

for i,x in enumerate(s):

if x not in d:

d[x]=1

else:

d[x]+=1

try:

while d['a']>0 and d['b']>0 and d['c']>0:

res+=len(s)-i

# print(s[left])

d[s[left]] -= 1

left += 1

except KeyError:

continue

return res

**1360. Number of Days Between Two Dates**

Write a program to count the number of days between two dates.

The two dates are given as strings, their format is YYYY-MM-DD as shown in the examples.

**Example 1:**

**Input:** date1 = "2019-06-29", date2 = "2019-06-30"

**Output:** 1

**My sol:**

from datetime import datetime

class Solution:

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

return abs((datetime.strptime(date2, '%Y-%m-%d').date() - datetime.strptime(date1, '%Y-%m-%d').date()).days)

**1365. How Many Numbers Are Smaller Than the Current Number**

Given the array nums, for each nums[i] find out how many numbers in the array are smaller than it. That is, for each nums[i] you have to count the number of valid j's such that j != i **and** nums[j] < nums[i].

Return the answer in an array.

**Example 1:**

**Input:** nums = [8,1,2,2,3]

**Output:** [4,0,1,1,3]

**Explanation:**

For nums[0]=8 there exist four smaller numbers than it (1, 2, 2 and 3).

For nums[1]=1 does not exist any smaller number than it.

For nums[2]=2 there exist one smaller number than it (1).

For nums[3]=2 there exist one smaller number than it (1).

For nums[4]=3 there exist three smaller numbers than it (1, 2 and 2).

**My sol:**

class Solution:

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

a=[]

for i in nums:

count =0

for j in nums:

if j <i:

count=count +1

a.append(count)

return(a)

**1370. Increasing Decreasing String**

You are given a string s. Reorder the string using the following algorithm:

1. Pick the **smallest** character from s and **append** it to the result.
2. Pick the **smallest** character from s which is greater than the last appended character to the result and **append** it.
3. Repeat step 2 until you cannot pick more characters.
4. Pick the **largest** character from s and **append** it to the result.
5. Pick the **largest** character from s which is smaller than the last appended character to the result and **append** it.
6. Repeat step 5 until you cannot pick more characters.
7. Repeat the steps from 1 to 6 until you pick all characters from s.

In each step, If the smallest or the largest character appears more than once you can choose any occurrence and append it to the result.

Return *the result string after sorting*s*with this algorithm*.

**Example 1:**

**Input:** s = "aaaabbbbcccc"

**Output:** "abccbaabccba"

**Explanation:** After steps 1, 2 and 3 of the first iteration, result = "abc"

After steps 4, 5 and 6 of the first iteration, result = "abccba"

First iteration is done. Now s = "aabbcc" and we go back to step 1

After steps 1, 2 and 3 of the second iteration, result = "abccbaabc"

After steps 4, 5 and 6 of the second iteration, result = "abccbaabccba"

**My sol:**

class Solution:

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

result = []

chars = 'abcdefghijklmnopqrstuvwxyz'

while s:

for c in chars:

if c in s:

result.append(c)

s = s.replace(c,'',1)

else:

chars = chars.replace(c,'')

chars = chars[::-1]

return ''.join(result)

**1374. Generate a String With Characters That Have Odd Counts**

Given an integer n, *return a string with n characters such that each character in such string occurs****an odd number of times***.

The returned string must contain only lowercase English letters. If there are multiples valid strings, return **any** of them.

**Example 1:**

**Input:** n = 4

**Output:** "pppz"

**Explanation:** "pppz" is a valid string since the character 'p' occurs three times and the character 'z' occurs once. Note that there are many other valid strings such as "ohhh" and "love".

**My sol:**

class Solution:

def generateTheString(self, n: int) -> str:

if n%2==0:

return (n-1)\*'a'+'b'

else:

return n\*'a'

**1379. Find a Corresponding Node of a Binary Tree in a Clone of That Tree**

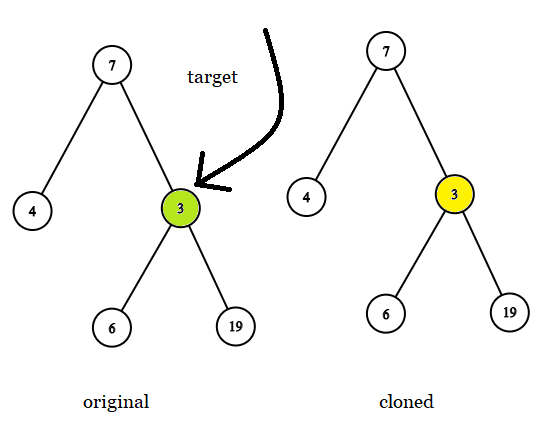
Given two binary trees original and cloned and given a reference to a node target in the original tree.

The cloned tree is a **copy of** the original tree.

Return *a reference to the same node* in the cloned tree.

**Note** that you are **not allowed** to change any of the two trees or the target node and the answer **must be** a reference to a node in the cloned tree.

**Example 1:**



**Input:** tree = [7,4,3,null,null,6,19], target = 3

**Output:** 3

**Explanation:** In all examples the original and cloned trees are shown. The target node is a green node from the original tree. The answer is the yellow node from the cloned tree.

**My sol:**

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

class Solution:

def getTargetCopy(self, original: TreeNode, cloned: TreeNode, target: TreeNode) -> TreeNode:

def temp(original,cloned):

if original:

if original is target:

self.ans=cloned

if original.left:

temp(original.left,cloned.left)

if original.right:

temp(original.right,cloned.right)

temp(original,cloned)

return self.ans

**1380. Lucky Numbers in a Matrix**

Given an m x n matrix of **distinct**numbers, return *all****lucky numbers****in the matrix in****any****order*.

A **lucky number** is an element of the matrix such that it is the minimum element in its row and maximum in its column.

**Example 1:**

**Input:** matrix = [[3,7,8],[9,11,13],[15,16,17]]

**Output:** [15]

**Explanation:** 15 is the only lucky number since it is the minimum in its row and the maximum in its column.

**My sol:**

import math

class Solution:

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

a=[]

b=[]

c=0

final=[]

for x in range(0,len(matrix)):

a.append(min(matrix[x]))

for x in range(0,len(matrix[0])):

b.append(max([row[c] for row in matrix]))

c+=1

print(a)

print(b)

for x in range(0,len(a)):

if a[x] in b:

final.append(a[x])

return final

**1385. Find the Distance Value Between Two Arrays**

Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

**Example 1:**

**Input:** arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2

**Output:** 2

**Explanation:**

For arr1[0]=4 we have:

|4-10|=6 > d=2

|4-9|=5 > d=2

|4-1|=3 > d=2

|4-8|=4 > d=2

For arr1[1]=5 we have:

|5-10|=5 > d=2

|5-9|=4 > d=2

|5-1|=4 > d=2

|5-8|=3 > d=2

For arr1[2]=8 we have:

**|8-10|=2 <= d=2**

**|8-9|=1 <= d=2**

|8-1|=7 > d=2

**|8-8|=0 <= d=2**

**My sol:**

class Solution:

def findTheDistanceValue(self, arr1: List[int], arr2: List[int], d: int) -> int:

count=0

flag=0

for x in arr1:

for y in arr2:

if abs(x-y)>d:

flag=0

else:

flag=1

break

if flag==0:

count=count+1

flag=0

return count

**1387. Sort Integers by The Power Value**

The power of an integer x is defined as the number of steps needed to transform x into 1 using the following steps:

* if x is even then x = x / 2
* if x is odd then x = 3 \* x + 1

For example, the power of x = 3 is 7 because 3 needs 7 steps to become 1 (3 --> 10 --> 5 --> 16 --> 8 --> 4 --> 2 --> 1).

Given three integers lo, hi and k. The task is to sort all integers in the interval [lo, hi] by the power value in **ascending order**, if two or more integers have **the same** power value sort them by **ascending order**.

Return the kth integer in the range [lo, hi] sorted by the power value.

Notice that for any integer x (lo <= x <= hi) it is **guaranteed** that x will transform into 1 using these steps and that the power of x is will **fit** in a 32-bit signed integer.

**Example 1:**

**Input:** lo = 12, hi = 15, k = 2

**Output:** 13

**Explanation:** The power of 12 is 9 (12 --> 6 --> 3 --> 10 --> 5 --> 16 --> 8 --> 4 --> 2 --> 1)

The power of 13 is 9

The power of 14 is 17

The power of 15 is 17

The interval sorted by the power value [12,13,14,15]. For k = 2 answer is the second element which is 13.

Notice that 12 and 13 have the same power value and we sorted them in ascending order. Same for 14 and 15.

**Example 2:**

**Input:** lo = 7, hi = 11, k = 4

**Output:** 7

**Explanation:** The power array corresponding to the interval [7, 8, 9, 10, 11] is [16, 3, 19, 6, 14].

The interval sorted by power is [8, 10, 11, 7, 9].

The fourth number in the sorted array is 7.

**My sol:**

class Solution:

def getKth(self, lo: int, hi: int, k: int) -> int:

count=0

dict={}

for x in range(lo,hi+1):

temp=x

count=0

while x!=1:

if x%2==0:

x=x/2

else:

x= 3\*x+1

count+=1

dict[temp]=count

ans={ke: v for ke, v in sorted(dict.items(), key=lambda item: (item[1],item[0]))}

c=1

for keys,v in ans.items():

if c==k:

return keys

c+=1

**1389. Create Target Array in the Given Order**

Given two arrays of integers nums and index. Your task is to create *target* array under the following rules:

* Initially *target* array is empty.
* From left to right read nums[i] and index[i], insert at index index[i] the value nums[i] in *target* array.
* Repeat the previous step until there are no elements to read in nums and index.

Return the *target* array.

It is guaranteed that the insertion operations will be valid.

**Example 1:**

**Input:** nums = [0,1,2,3,4], index = [0,1,2,2,1]

**Output:** [0,4,1,3,2]

**Explanation:**

nums index target

0 0 [0]

1 1 [0,1]

2 2 [0,1,2]

3 2 [0,1,3,2]

4 1 [0,4,1,3,2]

**My sol:**

class Solution:

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

target=[]

for i in range(0,len(nums)):

target.insert(index[i],nums[i])

return target

**1394. Find Lucky Integer in an Array**

Given an array of integers arr, a **lucky integer** is an integer that has a frequency in the array equal to its value.

Return *the largest****lucky integer****in the array*. If there is no **lucky integer** return -1.

**Example 1:**

**Input:** arr = [2,2,3,4]

**Output:** 2

**Explanation:** The only lucky number in the array is 2 because frequency[2] == 2.

**My sol:**

class Solution:

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

temp=[]

dic={}

for x in arr:

if x in dic:

dic[x]=dic[x]+1

else:

dic[x]=1

for key,value in dic.items():

if key==value:

temp.append(key)

if temp:

return max(temp)

else:

return -1

**1395. Count Number of Teams**

There are n soldiers standing in a line. Each soldier is assigned a **unique** rating value.

You have to form a team of 3 soldiers amongst them under the following rules:

* Choose 3 soldiers with index (i, j, k) with rating (rating[i], rating[j], rating[k]).
* A team is valid if: (rating[i] < rating[j] < rating[k]) or (rating[i] > rating[j] > rating[k]) where (0 <= i < j < k < n).

Return the number of teams you can form given the conditions. (soldiers can be part of multiple teams).

**Example 1:**

**Input:** rating = [2,5,3,4,1]

**Output:** 3

**Explanation:** We can form three teams given the conditions. (2,3,4), (5,4,1), (5,3,1).

**My sol:**

class Solution:

def numTeams(self, rating: List[int]) -> int:

# cache to store indices which have a value greater than current index

memoGreater = {}

# cache to store indices which have a value lesser than current index

memoLesser = {}

length = len(rating)

for i in range(length):

memoGreater[i] = []

memoLesser[i] = []

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

if rating[j] > rating[i]:

memoGreater[i].append(j)

elif rating[j] < rating[i]:

memoLesser[i].append(j)

print(memoGreater)

print(memoLesser)

numTeams = 0

# add the teams using the conditions specified

for i in range(0,len(rating)):

for index in memoGreater[i]:

numTeams += len(memoGreater[index])

for index in memoLesser[i]:

numTeams += len(memoLesser[index])

return numTeams

**1399. Count Largest Group**

You are given an integer n.

Each number from 1 to n is grouped according to the sum of its digits.

Return *the number of groups that have the largest size*.

**Example 1:**

**Input:** n = 13

**Output:** 4

**Explanation:** There are 9 groups in total, they are grouped according sum of its digits of numbers from 1 to 13:

[1,10], [2,11], [3,12], [4,13], [5], [6], [7], [8], [9].

There are 4 groups with largest size.

**My sol:**

class Solution:

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

a=[]

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

a.append(x)

sum=0

count=0

temp=[]

for x in a:

for y in str(x):

sum+=int(y)

temp.append(sum)

sum=0

c=collections.Counter(temp)

mval=-10000

for k,v in c.items():

# print(mval)

if v>mval:

mval=v

for k,v in c.items():

if v==mval:

count+=1

return count

1401-1450 ($$$$$ 15)

**1404. Number of Steps to Reduce a Number in Binary Representation to One**

Given the binary representation of an integer as a string s, return the number of steps to reduce it to 1 under the following rules:

* If the current number is even, you have to divide it by 2.
* If the current number is odd, you have to add 1 to it.

It is guaranteed that you can always reach one for all test cases.

**Example 1:**

**Input:** s = "1101"

**Output:** 6

**Explanation:** "1101" corressponds to number 13 in their decimal representation.

Step 1) 13 is odd, add 1 and obtain 14.

Step 2) 14 is even, divide by 2 and obtain 7.

Step 3) 7 is odd, add 1 and obtain 8.

Step 4) 8 is even, divide by 2 and obtain 4.

Step 5) 4 is even, divide by 2 and obtain 2.

Step 6) 2 is even, divide by 2 and obtain 1.

**My sol:**

class Solution:

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

a=int(s,2)

print(a)

count=0

while a!=1:

count+=1

if a%2==0:

a=a//2

else:

a=a+1

return count

**1408. String Matching in an Array**

Given an array of string words. Return all strings in words which is substring of another word in **any** order.

String words[i] is substring of words[j], if can be obtained removing some characters to left and/or right side of words[j].

**Example 1:**

**Input:** words = ["mass","as","hero","superhero"]

**Output:** ["as","hero"]

**Explanation:** "as" is substring of "mass" and "hero" is substring of "superhero".

["hero","as"] is also a valid answer.

**My sol:**

class Solution:

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

res=[]

for x in range(0,len(words)):

for y in range(0,len(words)):

if x !=y and words[x] in words[y] and words[x] not in res:

res.append(words[x])

return res

**1409. Queries on a Permutation With Key**

Given the array queries of positive integers between 1 and m, you have to process all queries[i] (from i=0 to i=queries.length-1) according to the following rules:

* In the beginning, you have the permutation P=[1,2,3,...,m].
* For the current i, find the position of queries[i] in the permutation P (**indexing from 0**) and then move this at the beginning of the permutation P. Notice that the position of queries[i] in P is the result for queries[i].

Return an array containing the result for the given queries.

**Example 1:**

**Input:** queries = [3,1,2,1], m = 5

**Output:** [2,1,2,1]

**Explanation:** The queries are processed as follow:

For i=0: queries[i]=3, P=[1,2,3,4,5], position of 3 in P is **2**, then we move 3 to the beginning of P resulting in P=[3,1,2,4,5].

For i=1: queries[i]=1, P=[3,1,2,4,5], position of 1 in P is **1**, then we move 1 to the beginning of P resulting in P=[1,3,2,4,5].

For i=2: queries[i]=2, P=[1,3,2,4,5], position of 2 in P is **2**, then we move 2 to the beginning of P resulting in P=[2,1,3,4,5].

For i=3: queries[i]=1, P=[2,1,3,4,5], position of 1 in P is **1**, then we move 1 to the beginning of P resulting in P=[1,2,3,4,5].

Therefore, the array containing the result is [2,1,2,1].

**My sol:**

class Solution:

def processQueries(self, queries: List[int], m: int) -> List[int]:

temp=[]

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

temp.append(i)

res=[]

for x in queries:

res.append(temp.index(x))

i=temp.index(x)

r=temp.pop(i)

temp=[r]+temp

return res

**1417. Reformat The String**

You are given an alphanumeric string s. (**Alphanumeric string** is a string consisting of lowercase English letters and digits).

You have to find a permutation of the string where no letter is followed by another letter and no digit is followed by another digit. That is, no two adjacent characters have the same type.

Return *the reformatted string* or return **an empty string** if it is impossible to reformat the string.

**Example 1:**

**Input:** s = "a0b1c2"

**Output:** "0a1b2c"

**Explanation:** No two adjacent characters have the same type in "0a1b2c". "a0b1c2", "0a1b2c", "0c2a1b" are also valid permutations.

**My sol:**

class Solution:

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

a=[]

b=[]

for x in range(0,len(s)):

if s[x] in ['0','1','2','3','4','5','6','7','8','9']:

a.append(s[x])

else:

b.append(s[x])

i=0

j=0

res=""

if len(s)==1:

return s

if len(a)==0 or len(b)==0:

return res

if abs(len(a)-len(b))>=2:

return res

while i!=len(a) and j!=len(b):

res+=a[i]

res+=b[j]

i+=1

j+=1

if len(a)>len(b):

res+=a[-1]

elif len(a)<len(b):

res=b[-1]+res

elif len(a)==len(b):

pass

return res

**1422. Maximum Score After Splitting a String**

Given a string s of zeros and ones, *return the maximum score after splitting the string into two****non-empty****substrings* (i.e. **left** substring and **right** substring).

The score after splitting a string is the number of **zeros** in the **left** substring plus the number of **ones** in the **right** substring.

**Example 1:**

**Input:** s = "011101"

**Output:** 5

**Explanation:**

All possible ways of splitting s into two non-empty substrings are:

left = "0" and right = "11101", score = 1 + 4 = 5

left = "01" and right = "1101", score = 1 + 3 = 4

left = "011" and right = "101", score = 1 + 2 = 3

left = "0111" and right = "01", score = 1 + 1 = 2

left = "01110" and right = "1", score = 2 + 1 = 3

**My sol:**

class Solution:

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

i=1

maxval=0

while i<len(s):

temp=s[:i].count('0')+s[i:].count('1')

maxval=max(maxval,temp)

i+=1

return maxval

**1424. Diagonal Traverse II**

Given a 2D integer array nums, return *all elements of*nums*in diagonal order as shown in the below images*.

**Example 1:**

Diagram

Description automatically generated with medium confidence

**Input:** nums = [[1,2,3],[4,5,6],[7,8,9]]

**Output:** [1,4,2,7,5,3,8,6,9]

My sol:

class Solution:

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

d = {}

for i in range(0,len(nums)):

for j in range(0,len(nums[i])):

if i+j in d:

d[i+j].append(nums[i][j])

else:

d[i+j] = [nums[i][j]]

print(d)

temp=[]

for k,v in d.items():

temp+=v[::-1]

return temp

**1426. Counting Elements**

Given an integer array arr, count how many elements x there are, such that x + 1 is also in arr. If there are duplicates in arr, count them separately.

**Example 1:**

**Input:** arr = [1,2,3]

**Output:** 2

**Explanation:** 1 and 2 are counted cause 2 and 3 are in arr.

**My sol:**

class Solution:

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

count=0

for x in arr:

if x+1 in arr:

count+=1

return count

**1427. Perform String Shifts ( left shift and right shift of strings logic)**

You are given a string s containing lowercase English letters, and a matrix shift, where shift[i] = [directioni, amounti]:

* directioni can be 0 (for left shift) or 1 (for right shift).
* amounti is the amount by which string s is to be shifted.
* A left shift by 1 means remove the first character of s and append it to the end.
* Similarly, a right shift by 1 means remove the last character of s and add it to the beginning.

Return the final string after all operations.

**Example 1:**

**Input:** s = "abc", shift = [[0,1],[1,2]]

**Output:** "cab"

**Explanation:**

[0,1] means shift to left by 1. "abc" -> "bca"

[1,2] means shift to right by 2. "bca" -> "cab"

**My sol:**

class Solution:

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

for x in shift:

if x[0]==0:

s=s[x[1]%len(s):]+s[:x[1]%len(s)]

if x[0]==1:

s=s[-x[1]%len(s):]+s[:-x[1]%len(s)]

return s

**1430. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree**

Given a binary tree where each path going from the root to any leaf form a **valid sequence**, check if a given string is a **valid sequence** in such binary tree.

We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a **sequence** in the given binary tree.

**Example 1:**

**A picture containing square

Description automatically generated**

**Input:** root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1]

**Output:** true

**Explanation:**

The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure).

Other valid sequences are:

0 -> 1 -> 1 -> 0

0 -> 0 -> 0

**My sol:**

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def isValidSequence(self, root: Optional[TreeNode], arr: List[int]) -> bool:

def temp(root,path):

if root:

path+=str(root.val)

if not root.left and not root.right:

paths.append(path)

else:

path+=''

temp(root.left,path)

temp(root.right,path)

paths=[]

temp(root,'')

print(paths)

s=''.join(str(x) for x in arr)

return s in paths

**1431. Kids With the Greatest Number of Candies**

There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have.

Return *a boolean array*result*of length*n*, where*result[i]*is*true*if, after giving the*ith*kid all the*extraCandies*, they will have the****greatest****number of candies among all the kids, or*false*otherwise*.

Note that **multiple** kids can have the **greatest** number of candies.

**Example 1:**

**Input:** candies = [2,3,5,1,3], extraCandies = 3

**Output:** [true,true,true,false,true]

**Explanation:** If you give all extraCandies to:

- Kid 1, they will have 2 + 3 = 5 candies, which is the greatest among the kids.

- Kid 2, they will have 3 + 3 = 6 candies, which is the greatest among the kids.

- Kid 3, they will have 5 + 3 = 8 candies, which is the greatest among the kids.

- Kid 4, they will have 1 + 3 = 4 candies, which is not the greatest among the kids.

- Kid 5, they will have 3 + 3 = 6 candies, which is the greatest among the kids.

**My sol:**

class Solution:

def kidsWithCandies(self, candies: List[int], extraCandies: int) -> List[bool]:

M=max(candies)

for x in range(0,len(candies)):

candies[x]=bool((candies[x]+extraCandies) >= M)

return candies

**1433. Check If a String Can Break Another String**

Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa.

A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.

**Example 1:**

**Input:** s1 = "abc", s2 = "xya"

**Output:** true

**Explanation:** "ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc".

**My sol:**

class Solution:

def checkIfCanBreak(self, s1: str, s2: str) -> bool:

if len(s1)!=len(s2):

return False

i=0

a=sorted(s1)

b=sorted(s2)

flag=0

temp=0

for x in range(0,len(s1)):

if a[x]>=b[x]:

flag=0

else:

flag=1

break

for x in range(0,len(s1)):

if a[x]<=b[x]:

temp=0

else:

temp=1

break

return flag==0 or temp==0

**1436. Destination City**

You are given the array paths, where paths[i] = [cityAi, cityBi] means there exists a direct path going from cityAi to cityBi. *Return the destination city, that is, the city without any path outgoing to another city.*

It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city.

**Example 1:**

**Input:** paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]

**Output:** "Sao Paulo"

**Explanation:** Starting at "London" city you will reach "Sao Paulo" city which is the destination city. Your trip consist of: "London" -> "New York" -> "Lima" -> "Sao Paulo".

**My sol:**

class Solution:

def destCity(self, paths: List[List[str]]) -> str:

a=[]

b=[]

for x in paths:

a.append(x[0])

b.append(x[1])

for x in b:

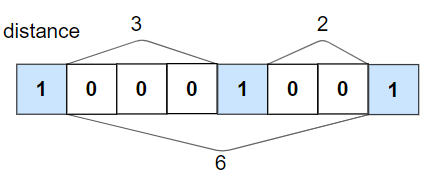
if x not in a:

return x

**1437. Check If All 1's Are at Least Length K Places Away**

Given an binary array nums and an integer k, return true*if all*1*'s are at least*k*places away from each other, otherwise return*false.

**Example 1:**



**Input:** nums = [1,0,0,0,1,0,0,1], k = 2

**Output:** true

**Explanation:** Each of the 1s are at least 2 places away from each other.

My sol:

class Solution:

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

a=[]

flag=0

for x in range(0,len(nums)):

if nums[x]==1:

a.append(x)

for y in range(1,len(a)):

if (a[y]-a[y-1])>k:

flag=0

else:

print(a[y])

flag=1

break

return flag==0

**1446. Consecutive Characters**

The **power** of the string is the maximum length of a non-empty substring that contains only one unique character.

Given a string s, return *the****power****of* s.

**Example 1:**

**Input:** s = "leetcode"

**Output:** 2

**Explanation:** The substring "ee" is of length 2 with the character 'e' only.

**My sol:**

class Solution:

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

count=1

result=1

for j in range(0,len(s)-1):

if s[j]==s[j+1]:

count+=1

else:

count=1

result=max(result,count)

return result

**1450. Number of Students Doing Homework at a Given Time**

Given two integer arrays startTime and endTime and given an integer queryTime.

The ith student started doing their homework at the time startTime[i] and finished it at time endTime[i].

Return *the number of students* doing their homework at time queryTime. More formally, return the number of students where queryTime lays in the interval [startTime[i], endTime[i]] inclusive.

**Example 1:**

**Input:** startTime = [1,2,3], endTime = [3,2,7], queryTime = 4

**Output:** 1

**Explanation:** We have 3 students where:

The first student started doing homework at time 1 and finished at time 3 and wasn't doing anything at time 4.

The second student started doing homework at time 2 and finished at time 2 and also wasn't doing anything at time 4.

The third student started doing homework at time 3 and finished at time 7 and was the only student doing homework at time 4.

1451-1500 ($$$$$ 16)

**1451. Rearrange Words in a Sentence**

Given a sentence text (A *sentence* is a string of space-separated words) in the following format:

* First letter is in upper case.
* Each word in text are separated by a single space.

Your task is to rearrange the words in text such that all words are rearranged in an increasing order of their lengths. If two words have the same length, arrange them in their original order.

Return the new text following the format shown above.

**Example 1:**

**Input:** text = "Leetcode is cool"

**Output:** "Is cool leetcode"

**Explanation:** There are 3 words, "Leetcode" of length 8, "is" of length 2 and "cool" of length 4.

Output is ordered by length and the new first word starts with capital letter.

**My sol:**

class Solution:

def arrangeWords(self, text: str) -> str:

a=text.split()

d={}

for x in range(0,len(a)):

d[x]=a[x].lower()

print(d)

dict={k: v for k, v in sorted(d.items(), key=lambda item: len(item[1]))}

print(dict)

res=[]

for x,y in dict.items():

res.append(y)

res[0]=res[0][0].upper()+res[0][1:]

return ' '.join(res)

**1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence**

Given a sentence that consists of some words separated by a **single space**, and a searchWord, check if searchWord is a prefix of any word in sentence.

Return *the index of the word in*sentence*(****1-indexed****) where*searchWord*is a prefix of this word*. If searchWord is a prefix of more than one word, return the index of the first word **(minimum index)**. If there is no such word return -1.

A **prefix** of a string s is any leading contiguous substring of s.

**Example 1:**

**Input:** sentence = "i love eating burger", searchWord = "burg"

**Output:** 4

**Explanation:** "burg" is prefix of "burger" which is the 4th word in the sentence.

**My sol:**

class Solution:

def isPrefixOfWord(self, sentence: str, searchWord: str) -> int:

arr=sentence.split()

for x in arr:

if searchWord==x[:len(searchWord)]:

return arr.index(x)+1

return -1

**1456. Maximum Number of Vowels in a Substring of Given Length**

Given a string s and an integer k, return *the maximum number of vowel letters in any substring of*s*with length*k.

**Vowel letters** in English are 'a', 'e', 'i', 'o', and 'u'.

**Example 1:**

**Input:** s = "abciiidef", k = 3

**Output:** 3

**Explanation:** The substring "iii" contains 3 vowel letters.

**My sol:**

class Solution:

def maxVowels(self, s: str, k: int) -> int:

i=0

j=0

res=0

while j+k<len(s)+1:

temp=s[i:j+k].count('a')+s[i:j+k].count('e')+s[i:j+k].count('i')+s[i:j+k].count('o')+s[i:j+k].count('u')

res=max(temp,res)

i+=1

j+=1

return res

**1460. Make Two Arrays Equal by Reversing Sub-arrays**

You are given two integer arrays of equal length target and arr. In one step, you can select any **non-empty sub-array** of arr and reverse it. You are allowed to make any number of steps.

Return true *if you can make*arr*equal to*target*or*false*otherwise*.

**Example 1:**

**Input:** target = [1,2,3,4], arr = [2,4,1,3]

**Output:** true

**Explanation:** You can follow the next steps to convert arr to target:

1- Reverse sub-array [2,4,1], arr becomes [1,4,2,3]

2- Reverse sub-array [4,2], arr becomes [1,2,4,3]

3- Reverse sub-array [4,3], arr becomes [1,2,3,4]

There are multiple ways to convert arr to target, this is not the only way to do so.

**My sol:**

class Solution:

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

if sorted(target)==sorted(arr):

return 1==1

else:

return 1==2

**1461. Check If a String Contains All Binary Codes of Size K**

Given a binary string s and an integer k, return true *if every binary code of length* k *is a substring of* s. Otherwise, return false.

**Example 1:**

**Input:** s = "00110110", k = 2

**Output:** true

**Explanation:** The binary codes of length 2 are "00", "01", "10" and "11". They can be all found as substrings at indices 0, 1, 3 and 2 respectively.

**My sol:**

class Solution:

def hasAllCodes(self, s: str, k: int) -> bool:

temp=set()

for x in range(0,len(s)-k+1):

temp.add(s[x:x+k])

return len(temp)==pow(2,k)

**1464. Maximum Product of Two Elements in an Array**

Given the array of integers nums, you will choose two different indices i and j of that array. *Return the maximum value of* (nums[i]-1)\*(nums[j]-1).

**Example 1:**

**Input:** nums = [3,4,5,2]

**Output:** 12

**Explanation:** If you choose the indices i=1 and j=2 (indexed from 0), you will get the maximum value, that is, (nums[1]-1)\*(nums[2]-1) = (4-1)\*(5-1) = 3\*4 = 12.

**My sol:**

class Solution:

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

a=max(nums)

nums.remove(a)

print(nums)

b=max(nums)

nums.remove(b)

return (a-1) \*(b-1)

**1469. Find All The Lonely Nodes**

In a binary tree, a **lonely** node is a node that is the only child of its parent node. The root of the tree is not lonely because it does not have a parent node.

Given the root of a binary tree, return *an array containing the values of all lonely nodes* in the tree. Return the list **in any order**.

**Example 1:**

A picture containing text, pool ball, gauge

Description automatically generated

**Input:** root = [1,2,3,null,4]

**Output:** [4]

**Explanation:** Light blue node is the only lonely node.

Node 1 is the root and is not lonely.

Nodes 2 and 3 have the same parent and are not lonely.

**My sol:**

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

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

a=[]

def temp(root):

if not root:

pass

if root.left:

if not root.right:

a.append(root.left.val)

temp(root.left)

if root.right:

if not root.left:

a.append(root.right.val)

temp(root.right)

temp(root)

return a

**1470. Shuffle the Array**

Given the array nums consisting of 2n elements in the form [x1,x2,...,xn,y1,y2,...,yn].

*Return the array in the form* [x1,y1,x2,y2,...,xn,yn].

**Example 1:**

**Input:** nums = [2,5,1,3,4,7], n = 3

**Output:** [2,3,5,4,1,7]

**Explanation:** Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].

**My soL:**

class Solution:

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

l= []

a = nums[0:n]

b = nums[n:]

for i in range(n):

l.append(a[i])

l.append(b[i])

return l

**1474. Delete N Nodes After M Nodes of a Linked List**

You are given the head of a linked list and two integers m and n.

Traverse the linked list and remove some nodes in the following way:

* Start with the head as the current node.
* Keep the first m nodes starting with the current node.
* Remove the next n nodes
* Keep repeating steps 2 and 3 until you reach the end of the list.

Return *the head of the modified list after removing the mentioned nodes*.

**Example 1:**

Diagram

Description automatically generated

**Input:** head = [1,2,3,4,5,6,7,8,9,10,11,12,13], m = 2, n = 3

**Output:** [1,2,6,7,11,12]

**Explanation:** Keep the first (m = 2) nodes starting from the head of the linked List (1 ->2) show in black nodes.

Delete the next (n = 3) nodes (3 -> 4 -> 5) show in read nodes.

Continue with the same procedure until reaching the tail of the Linked List.

Head of the linked list after removing nodes is returned.

My sol:

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def deleteNodes(self, head: ListNode, m: int, n: int) -> ListNode:

dummy=ListNode()

p=dummy

cur=head

temp=[]

res=[]

while cur:

temp.append(cur.val)

cur=cur.next

a=0

res=[]

while a+n<=len(temp):

res+=temp[a:a+m]

a+=m+n

res+=temp[a:][:m]

for x in res:

p.next=ListNode(x)

p=p.next

return dummy.next

**1475. Final Prices With a Special Discount in a Shop**

Given the array prices where prices[i] is the price of the ith item in a shop. There is a special discount for items in the shop, if you buy the ith item, then you will receive a discount equivalent to prices[j] where j is the **minimum** index such that j > i and prices[j] <= prices[i], otherwise, you will not receive any discount at all.

*Return an array where the ith element is the final price you will pay for the ith item of the shop considering the special discount.*

**Example 1:**

**Input:** prices = [8,4,6,2,3]

**Output:** [4,2,4,2,3]

**Explanation:**

For item 0 with price[0]=8 you will receive a discount equivalent to prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4.

For item 1 with price[1]=4 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2.

For item 2 with price[2]=6 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4.

For items 3 and 4 you will not receive any discount at all.

**My sol:**

class Solution:

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

for i in range(0,len(prices)):

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

if j>i and prices[j]<= prices[i]:

prices[i]=prices[i]-prices[j]

break

return prices

**1480. Running Sum of 1d Array**

Given an array nums. We define a running sum of an array as runningSum[i] = sum(nums[0]…nums[i]).

Return the running sum of nums.

**Example 1:**

**Input:** nums = [1,2,3,4]

**Output:** [1,3,6,10]

**Explanation:** Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].

**My sol:**

class Solution:

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

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

nums[x]=nums[x]+nums[x-1]

return nums

**1481. Least Number of Unique Integers after K Removals**

Given an array of integers arr and an integer k. Find the *least number of unique integers* after removing **exactly** k elements**.**

**Example 1:**

**Input:** arr = [5,5,4], k = 1

**Output:** 1

**Explanation**: Remove the single 4, only 5 is left.

**My sol:**

class Solution:

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

a=collections.Counter(arr)

b=dict(sorted(a.items(), key=lambda item: item[1]))

res=0

for key,value in b.items():

while k!=0 and b[key]>0:

b[key]=b[key]-1

k-=1

for key, value in b.items():

if value>0:

res+=1

return res

**1486. XOR Operation in an Array**

You are given an integer n and an integer start.

Define an array nums where nums[i] = start + 2 \* i (**0-indexed**) and n == nums.length.

Return *the bitwise XOR of all elements of* nums.

**Example 1:**

**Input:** n = 5, start = 0

**Output:** 8

**Explanation:** Array nums is equal to [0, 2, 4, 6, 8] where (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8.

Where "^" corresponds to bitwise XOR operator.

**My sol:**

class Solution:

def xorOperation(self, n: int, start: int) -> int:

nums=[]

sum=0

target=0

for i in range(0,n):

nums.insert(i,(start + 2 \*i))

print(nums)

#for j in range(0,(len(nums)-1)):

j=0

while(j!=n):

sum=sum ^ nums[j]

#print('\*')

j=j+1

return sum

**1490. Clone N-ary Tree**

Given a root of an N-ary tree, return a [**deep copy**](https://en.wikipedia.org/wiki/Object_copying#Deep_copy) (clone) of the tree.

Each node in the n-ary tree contains a val (int) and a list (List[Node]) of its children.

class Node {

public int val;

public List<Node> children;

}

*Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).*

**Example 1:**

Shape

Description automatically generated

**Input:** root = [1,null,3,2,4,null,5,6]

**Output:** [1,null,3,2,4,null,5,6]

**My sol:**

"""

# Definition for a Node.

class Node:

def \_\_init\_\_(self, val=None, children=None):

self.val = val

self.children = children if children is not None else []

"""

class Solution:

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

if not root:

return None

node = Node(root.val)

for child in root.children:

node.children.append(self.cloneTree(child))

return node

**1491. Average Salary Excluding the Minimum and Maximum Salary**

You are given an array of **unique** integers salary where salary[i] is the salary of the ith employee.

Return *the average salary of employees excluding the minimum and maximum salary*. Answers within 10-5 of the actual answer will be accepted.

**Example 1:**

**Input:** salary = [4000,3000,1000,2000]

**Output:** 2500.00000

**Explanation:** Minimum salary and maximum salary are 1000 and 4000 respectively.

Average salary excluding minimum and maximum salary is (2000+3000) / 2 = 2500

**My sol:**

class Solution:

def average(self, salary: List[int]) -> float:

salary.remove(max(salary))

salary.remove(min(salary))

sum=0

for i in salary:

sum=sum+i

average=sum/len(salary)

return average

**1492. The kth Factor of n**

You are given two positive integers n and k. A factor of an integer n is defined as an integer i where n % i == 0.

Consider a list of all factors of n sorted in **ascending order**, return *the*kth*factor* in this list or return -1 if n has less than k factors.

**Example 1:**

**Input:** n = 12, k = 3

**Output:** 3

**Explanation:** Factors list is [1, 2, 3, 4, 6, 12], the 3rd factor is 3.

**My sol:**

class Solution:

def kthFactor(self, n: int, k: int) -> int:

c=0

final=-1

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

if n%i==0:

c+=1

if c>=k:

final=i

break

if c==k:

return final

if c<k:

return -1

1501-1550 ($$$$$ 12)

**1502. Can Make Arithmetic Progression From Sequence**

A sequence of numbers is called an **arithmetic progression** if the difference between any two consecutive elements is the same.

Given an array of numbers arr, return true *if the array can be rearranged to form an****arithmetic progression****. Otherwise, return* false.

**Example 1:**

**Input:** arr = [3,5,1]

**Output:** true

**Explanation:** We can reorder the elements as [1,3,5] or [5,3,1] with differences 2 and -2 respectively, between each consecutive elements.

**My sol:**

class Solution:

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

s = sorted(arr)

diff = s[0] - s[1]

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

if diff != s[i - 1] - s[i]:

return False

return True

**1507. Reformat Date**

Given a date string in the form Day Month Year, where:

* Day is in the set {"1st", "2nd", "3rd", "4th", ..., "30th", "31st"}.
* Month is in the set {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"}.
* Year is in the range [1900, 2100].

Convert the date string to the format YYYY-MM-DD, where:

* YYYY denotes the 4 digit year.
* MM denotes the 2 digit month.
* DD denotes the 2 digit day.

**Example 1:**

**Input:** date = "20th Oct 2052"

**Output:** "2052-10-20"

**My sol:**

class Solution:

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

month=""

day=""

year=""

d=[]

d=date.split()

year=d[2]

month=d[1]

if d[1]=="Jan":

month="01"

elif d[1]=="Feb":

month="02"

elif d[1]=="Mar":

month="03"

elif d[1]=="Apr":

month="04"

elif d[1]=="May":

month="05"

elif d[1]=="Jun":

month="06"

elif d[1]=="Jul":

month="07"

elif d[1]=="Aug":

month="08"

elif d[1]=="Sep":

month="09"

elif d[1]=="Oct":

month="10"

elif d[1]=="Nov":

month="11"

elif d[1]=="Dec":

month="12"

ddup=d[0]

day=ddup[0:-2]

#print(date)

if int(day)<10:

day="0"+day

else:

pass

return year+"-"+month+"-"+day

**1512. Number of Good Pairs**

Given an array of integers nums, return *the number of****good pairs***.

A pair (i, j) is called *good* if nums[i] == nums[j] and i < j.

**Example 1:**

**Input:** nums = [1,2,3,1,1,3]

**Output:** 4

**Explanation:** There are 4 good pairs (0,3), (0,4), (3,4), (2,5) 0-indexed.

**My sol:**

class Solution:

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

a=[]

for i in range(0,len(nums)):

for j in range(0,len(nums)):

if i<j and (nums[i] == nums[j]):

a.append([i,j])

return len(a)

**1513. Number of Substrings With Only 1s**

Given a binary string s, return *the number of substrings with all characters* 1*'s*. Since the answer may be too large, return it modulo 109 + 7.

**Example 1:**

**Input:** s = "0110111"

**Output:** 9

**Explanation:** There are 9 substring in total with only 1's characters.

"1" -> 5 times.

"11" -> 3 times.

"111" -> 1 time.

**My sol:**

class Solution:

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

res=0

temp=0

for x in s:

if x=='0':

temp=0

else:

temp+=1

res+=temp

return res % ((10\*\*9)+7)

**1518. Water Bottles**

There are numBottles water bottles that are initially full of water. You can exchange numExchange empty water bottles from the market with one full water bottle.

The operation of drinking a full water bottle turns it into an empty bottle.

Given the two integers numBottles and numExchange, return *the****maximum****number of water bottles you can drink*.

**Example 1:**

Diagram

Description automatically generated

**Input:** numBottles = 9, numExchange = 3

**Output:** 13

**Explanation:** You can exchange 3 empty bottles to get 1 full water bottle.

Number of water bottles you can drink: 9 + 3 + 1 = 13.

**My sol:**

class Solution:

def numWaterBottles(self, numBottles: int, numExchange: int) -> int:

count=numBottles

while numBottles>numExchange:

if numBottles%numExchange==0:

count+=numBottles//numExchange

numBottles=numBottles//numExchange

if numBottles%numExchange!=0:

count+=numBottles//numExchange

numBottles=(numBottles%numExchange)+(numBottles//numExchange)

return count+numBottles//numExchange

**1523. Count Odd Numbers in an Interval Range**

Given two non-negative integers low and high. Return the *count of odd numbers between*low*and*high*(inclusive)*.

**Example 1:**

**Input:** low = 3, high = 7

**Output:** 3

**Explanation:** The odd numbers between 3 and 7 are [3,5,7].

**My sol:**

class Solution:

def countOdds(self, low: int, high: int) -> int:

if high%2==0 and low%2==0:

return int((high-low)/2)

else:

return int((high-low)/2)+1

**1525. Number of Good Ways to Split a String**

You are given a string s.

A split is called **good** if you can split s into two non-empty strings sleft and sright where their concatenation is equal to s (i.e., sleft + sright = s) and the number of distinct letters in sleft and sright is the same.

Return *the number of****good splits****you can make in s*.

**Example 1:**

**Input:** s = "aacaba"

**Output:** 2

**Explanation:** There are 5 ways to split "aacaba" and 2 of them are good.

("a", "acaba") Left string and right string contains 1 and 3 different letters respectively.

("aa", "caba") Left string and right string contains 1 and 3 different letters respectively.

("aac", "aba") Left string and right string contains 2 and 2 different letters respectively (good split).

("aaca", "ba") Left string and right string contains 2 and 2 different letters respectively (good split).

("aacab", "a") Left string and right string contains 3 and 1 different letters respectively.

**My sol:**

class Solution:

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

# i=1

# count=0

# while i <len(s):

# if len(list(set(s[:i])))==len(list(set(s[i:]))):

# count+=1

# i+=1

# return count

left\_count = collections.Counter()

right\_count = collections.Counter(s)

res = 0

for c in s:

left\_count[c] += 1

right\_count[c] -= 1

if right\_count[c] == 0:

del right\_count[c]

if len(left\_count) == len(right\_count):

res += 1

return res

**1528. Shuffle String**

You are given a string s and an integer array indices of the **same length**. The string s will be shuffled such that the character at the ith position moves to indices[i] in the shuffled string.

Return *the shuffled string*.

**Example 1:**

Diagram

Description automatically generated with medium confidence

**Input:** s = "codeleet", indices = [4,5,6,7,0,2,1,3]

**Output:** "leetcode"

**Explanation:** As shown, "codeleet" becomes "leetcode" after shuffling.

**My sol:**

class Solution:

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

final=""

ans=[" " for x in range(0,len(indices))]

for ch in range(0,len(indices)):

ans[indices[ch]] = s[ch]

final="".join(ans)

return final

**1534. Count Good Triplets**

Given an array of integers arr, and three integers a, b and c. You need to find the number of good triplets.

A triplet (arr[i], arr[j], arr[k]) is **good** if the following conditions are true:

* 0 <= i < j < k < arr.length
* |arr[i] - arr[j]| <= a
* |arr[j] - arr[k]| <= b
* |arr[i] - arr[k]| <= c

Where |x| denotes the absolute value of x.

Return*the number of good triplets*.

**Example 1:**

**Input:** arr = [3,0,1,1,9,7], a = 7, b = 2, c = 3

**Output:** 4

**Explanation:** There are 4 good triplets: [(3,0,1), (3,0,1), (3,1,1), (0,1,1)].

**My sol:**

class Solution:

def countGoodTriplets(self, arr: List[int], a: int, b: int, c: int) -> int:

count=0

for x in range(0,len(arr)):

for y in range(0,len(arr)):

for z in range(0,len(arr)):

if x!=y and y!=z and z!=x and x<y and y<z and abs(arr[x]-arr[y])<=a and abs(arr[y]-arr[z])<=b and abs(arr[z]-arr[x])<=c:

count+=1

return count

**1539. Kth Missing Positive Number (kiran amazon question)**

Given an array arr of positive integers sorted in a **strictly increasing order**, and an integer k.

Return *the* kth ***positive****integer that is****missing****from this array.*

**Example 1:**

**Input:** arr = [2,3,4,7,11], k = 5

**Output:** 9

**Explanation:** The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th missing positive integer is 9.

**My sol:**

class Solution:

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

p=list(range(1,3000))

a=list(set(p)-set(arr))

return a[k-1]

**optimized binary search solution ( o(log n)):**

class Solution:

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

left, right = 0, len(arr) - 1

while left <= right:

pivot = (left + right) // 2

# If number of positive integers

# which are missing before arr[pivot]

# is less than k -->

# continue to search on the right.

if arr[pivot] - pivot - 1 < k:

left = pivot + 1

# Otherwise, go left.

else:

right = pivot - 1

# At the end of the loop, left = right + 1,

# and the kth missing is in-between arr[right] and arr[left].

# The number of integers missing before arr[right] is

# arr[right] - right - 1 -->

# the number to return is

# arr[right] + k - (arr[right] - right - 1) = k + left

return left + k

**1544. Make The String Great**

Given a string s of lower and upper case English letters.

A good string is a string which doesn't have **two adjacent characters** s[i] and s[i + 1] where:

* 0 <= i <= s.length - 2
* s[i] is a lower-case letter and s[i + 1] is the same letter but in upper-case or **vice-versa**.

To make the string good, you can choose **two adjacent** characters that make the string bad and remove them. You can keep doing this until the string becomes good.

Return *the string* after making it good. The answer is guaranteed to be unique under the given constraints.

**Notice** that an empty string is also good.

**Example 1:**

**Input:** s = "leEeetcode"

**Output:** "leetcode"

**Explanation:** In the first step, either you choose i = 1 or i = 2, both will result "leEeetcode" to be reduced to "leetcode".

**My sol:**

class Solution:

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

stack=[]

# ch=""

for i in s:

if stack and i.upper()==stack[-1] and i!=stack[-1]:

stack.pop()

# ch=ch[:-1]

elif stack and stack[-1].upper()==i and i!=stack[-1]:

stack.pop()

else:

stack.append(i)

# ch+=i

ans="".join(stack)

return ans

**1550. Three Consecutive Odds**

Given an integer array arr, return true if there are three consecutive odd numbers in the array. Otherwise, return false.

**Example 1:**

**Input:** arr = [2,6,4,1]

**Output:** false

**Explanation:** There are no three consecutive odds.

**My sol:**

class Solution:

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

flag=0

for i in range(0,len(arr)-2):

if arr[i] %2 != 0 and arr[i+1] % 2 != 0 and arr[i+2] %2 != 0:

flag=1

break

else:

flag=0

if len(arr)==1:

return 1==2

elif len(arr)==2:

return 1==2

else:

return flag==1

1551-1600 ($$$$$ 11)

**1551. Minimum Operations to Make Array Equal**

You have an array arr of length n where arr[i] = (2 \* i) + 1 for all valid values of i (i.e., 0 <= i < n).

In one operation, you can select two indices x and y where 0 <= x, y < n and subtract 1 from arr[x] and add 1 to arr[y] (i.e., perform arr[x] -=1 and arr[y] += 1). The goal is to make all the elements of the array **equal**. It is **guaranteed** that all the elements of the array can be made equal using some operations.

Given an integer n, the length of the array, return *the minimum number of operations* needed to make all the elements of arr equal.

**Example 1:**

**Input:** n = 3

**Output:** 2

**Explanation:** arr = [1, 3, 5]

First operation choose x = 2 and y = 0, this leads arr to be [2, 3, 4]

In the second operation choose x = 2 and y = 0 again, thus arr = [3, 3, 3].

**My sol:**

class Solution:

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

arr=[]

i=0

while i<n:

arr.append(2\*i+1)

i+=1

temp=sum(arr)

count=0

for x in arr:

if (temp//len(arr))-x>0:

count+=(temp//len(arr))-x

return count

**1556. Thousand Separator**

Given an integer n, add a dot (".") as the thousands separator and return it in string format.

**Example 1:**

**Input:** n = 987

**Output:** "987"

**My sol:**

class Solution:

def thousandSeparator(self, n: int) -> str:

if len(str(n))<=3:

return str(n)

a=str(n)[::-1]

k=3

res=""

while k<len(a):

res=a[:k]+'.'+a[k:]

k+=4

a=res

return res[::-1]

**1561. Maximum Number of Coins You Can Get**

There are 3n piles of coins of varying size, you and your friends will take piles of coins as follows:

* In each step, you will choose **any**3 piles of coins (not necessarily consecutive).
* Of your choice, Alice will pick the pile with the maximum number of coins.
* You will pick the next pile with the maximum number of coins.
* Your friend Bob will pick the last pile.
* Repeat until there are no more piles of coins.

Given an array of integers piles where piles[i] is the number of coins in the ith pile.

Return the maximum number of coins that you can have.

**Example 1:**

**Input:** piles = [2,4,1,2,7,8]

**Output:** 9

**Explanation:** Choose the triplet (2, 7, 8), Alice Pick the pile with 8 coins, you the pile with **7** coins and Bob the last one.

Choose the triplet (1, 2, 4), Alice Pick the pile with 4 coins, you the pile with **2** coins and Bob the last one.

The maximum number of coins which you can have are: 7 + 2 = 9.

On the other hand if we choose this arrangement (1, **2**, 8), (2, **4**, 7) you only get 2 + 4 = 6 coins which is not optimal.

**My sol:**

class Solution:

def maxCoins(self, piles: List[int]) -> int:

count=0

i=0

temp=0

if len(piles)==3:

return sorted(piles)[1]

A=sorted(piles)[::-1]

while i<len(piles)-3:

temp+=1

count+=A[i+1]

if temp>=(len(piles)//3):

return count

i+=2

return count

**1566. Detect Pattern of Length M Repeated K or More Times**

Given an array of positive integers arr, find a pattern of length m that is repeated k or more times.

A **pattern** is a subarray (consecutive sub-sequence) that consists of one or more values, repeated multiple times **consecutively**without overlapping. A pattern is defined by its length and the number of repetitions.

Return true *if there exists a pattern of length* m *that is repeated* k *or more times, otherwise return* false.

**Example 1:**

**Input:** arr = [1,2,4,4,4,4], m = 1, k = 3

**Output:** true

**Explanation:** The pattern **(4)** of length 1 is repeated 4 consecutive times. Notice that pattern can be repeated k or more times but not less.

**My sol:**

class Solution:

def containsPattern(self, arr: List[int], m: int, k: int) -> bool:

i=0

j=0

s=''.join(str(x) for x in arr)

while i<=len(arr):

while j<i and len(arr[j:i])==m:

if len(arr[j:i])==1 and str(arr[j:i][0])\*k in s and arr.count(arr[j:i][0])>=k:

return True

if len(arr[j:i])!=1 and s.count(''.join(str(x) for x in arr[j:i]))>=k and ''.join(str(x) for x in arr[j:i])\*k in s:

return True

j+=1

i+=1

return False

**1570. Dot Product of Two Sparse Vectors**

Given two sparse vectors, compute their dot product.

Implement class SparseVector:

* SparseVector(nums) Initializes the object with the vector nums
* dotProduct(vec) Compute the dot product between the instance of *SparseVector* and vec

A **sparse vector** is a vector that has mostly zero values, you should store the sparse vector **efficiently**and compute the dot product between two *SparseVector*.

**Follow up:**What if only one of the vectors is sparse?

**Example 1:**

**Input:** nums1 = [1,0,0,2,3], nums2 = [0,3,0,4,0]

**Output:** 8

**Explanation:** v1 = SparseVector(nums1) , v2 = SparseVector(nums2)

v1.dotProduct(v2) = 1\*0 + 0\*3 + 0\*0 + 2\*4 + 3\*0 = 8

**My sol:**

import numpy as np

class SparseVector:

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

self.array = nums

# Return the dotProduct of two sparse vectors

def dotProduct(self, vec: 'SparseVector') -> int:

result = 0

for num1, num2 in zip(self.array, vec.array):

result += num1 \* num2

return result

# Your SparseVector object will be instantiated and called as such:

# v1 = SparseVector(nums1)

# v2 = SparseVector(nums2)

# ans = v1.dotProduct(v2)

**1572. Matrix Diagonal Sum**

Given a square matrix mat, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

**Example 1:**

Diagram

Description automatically generated with medium confidence

**Input:** mat = [[**1**,2,**3**],

  [4,**5**,6],

  [**7**,8,**9**]]

**Output:** 25

**Explanation:** Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25

Notice that element mat[1][1] = 5 is counted only once.

**My sol:**

class Solution:

def diagonalSum(self, mat: List[List[int]]) -> int:

sum=0

for x in range(0,len(mat)):

for y in range(0,len(mat[0])):

if x==y:

print(mat[x][y])

sum+=mat[x][y]

if x+y==len(mat)-1 and x!=y:

print(mat[x][y])

sum+=mat[x][y]

return sum

**1576. Replace All ?'s to Avoid Consecutive Repeating Characters**

Given a string s containing only lowercase English letters and the '?' character, convert **all**the '?' characters into lowercase letters such that the final string does not contain any **consecutive repeating**characters. You **cannot**modify the non '?' characters.

It is **guaranteed**that there are no consecutive repeating characters in the given string **except**for '?'.

Return *the final string after all the conversions (possibly zero) have been made*. If there is more than one solution, return **any of them**. It can be shown that an answer is always possible with the given constraints.

**Example 1:**

**Input:** s = "?zs"

**Output:** "azs"

**Explanation:** There are 25 solutions for this problem. From "azs" to "yzs", all are valid. Only "z" is an invalid modification as the string will consist of consecutive repeating characters in "zzs".

**My sol:**

import random

class Solution:

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

arr=['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']

ans=[]

for temp in arr:

if temp not in s:

ans.append(temp)

i=0

for x in range(0,len(s)):

if s[x]=='?':

s=s.replace(s[x],ans[i],1)

if i==len(ans)-1:

i=0

i+=1

if i==len(s)-1:

i=0

for x in range(0,len(s)-1):

if s[x]==s[x+1]:

s=s.replace(s[x],ans,1)

i+=1

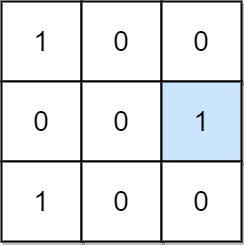
return s

**1582. Special Positions in a Binary Matrix ( matrix column logic)**

Given an m x n binary matrix mat, return *the number of special positions in*mat*.*

A position (i, j) is called **special** if mat[i][j] == 1 and all other elements in row i and column j are 0 (rows and columns are **0-indexed**).

**Example 1:**



**Input:** mat = [[1,0,0],[0,0,1],[1,0,0]]

**Output:** 1

**Explanation:** (1, 2) is a special position because mat[1][2] == 1 and all other elements in row 1 and column 2 are 0.

**My sol:**

class Solution:

def numSpecial(self, mat: List[List[int]]) -> int:

s=""

i=0

b=[]

a=[]

res=0

while i<len(mat[0]):

for x in mat:

a.append(x[i])

b.append(a)

a=[]

i+=1

print(b)

for x in range(0,len(mat)):

for y in range(0,len(mat[0])):

if mat[x][y]==1 and mat[x].count(1)==1 and b[y].count(1)==1:

res+=1

return res

**1588. Sum of All Odd Length Subarrays**

Given an array of positive integers arr, return *the sum of all possible****odd-length subarrays****of*arr.

A **subarray** is a contiguous subsequence of the array.

**Example 1:**

**Input:** arr = [1,4,2,5,3]

**Output:** 58

**Explanation:** The odd-length subarrays of arr and their sums are:

[1] = 1

[4] = 4

[2] = 2

[5] = 5

[3] = 3

[1,4,2] = 7

[4,2,5] = 11

[2,5,3] = 10

[1,4,2,5,3] = 15

If we add all these together we get 1 + 4 + 2 + 5 + 3 + 7 + 11 + 10 + 15 = 58

**My sol:**

import itertools

class Solution:

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

count=0

def allSubArrays(xs):

n = len(xs)

indices = list(range(n+1))

for i,j in itertools.combinations(indices,2):

yield xs[i:j]

for x in list(allSubArrays(arr)):

if len(x)%2!=0:

count+=sum(x)

return count

**1592. Rearrange Spaces Between Words**

You are given a string text of words that are placed among some number of spaces. Each word consists of one or more lowercase English letters and are separated by at least one space. It's guaranteed that text **contains at least one word**.

Rearrange the spaces so that there is an **equal** number of spaces between every pair of adjacent words and that number is **maximized**. If you cannot redistribute all the spaces equally, place the **extra spaces at the end**, meaning the returned string should be the same length as text.

Return *the string after rearranging the spaces*.

**Example 1:**

**Input:** text = " this is a sentence "

**Output:** "this is a sentence"

**Explanation:** There are a total of 9 spaces and 4 words. We can evenly divide the 9 spaces between the words: 9 / (4-1) = 3 spaces.

**My sol:**

class Solution:

def reorderSpaces(self, text: str) -> str:

a=text.count(" ")

b=text.split()

s=""

if len(text)==1:

return text

if len(b)==1:

s+=b[0]+(" "\*a)

return s

c=a//(len(b)-1)

rem=a%(len(b)-1)

for x in range(0,len(b)):

c=a//(len(b)-1)

s+=b[x]

if x==len(b)-1:

break

while c>0:

s+=" "

c-=1

while rem>0:

s+=" "

rem-=1

return s

**1598. Crawler Log Folder**

The Leetcode file system keeps a log each time some user performs a *change folder* operation.

The operations are described below:

* "../" : Move to the parent folder of the current folder. (If you are already in the main folder, **remain in the same folder**).
* "./" : Remain in the same folder.
* "x/" : Move to the child folder named x (This folder is **guaranteed to always exist**).

You are given a list of strings logs where logs[i] is the operation performed by the user at the ith step.

The file system starts in the main folder, then the operations in logs are performed.

Return *the minimum number of operations needed to go back to the main folder after the change folder operations.*

**Example 1:**

Chart, diagram, box and whisker chart

Description automatically generated

**Input:** logs = ["d1/","d2/","../","d21/","./"]

**Output:** 2

**Explanation:** Use this change folder operation "../" 2 times and go back to the main folder.

**My sol:**

class Solution:

def minOperations(self, logs: List[str]) -> int:

stack=[]

for x in logs:

if stack and x=="../":

stack.pop()

elif stack and x=="./":

pass

elif x=="../" and not stack:

pass

elif x=="./" and not stack:

pass

else:

stack.append(x)

return len(stack)

1601-1650 ($$$$$ 9)

**1602. Find Nearest Right Node in Binary Tree**

Given the root of a binary tree and a node u in the tree, return *the****nearest****node on the****same level****that is to the****right****of* u*, or return* null *if*u *is the rightmost node in its level*.

**Example 1:**

A picture containing text, pool ball, night sky

Description automatically generated

**Input:** root = [1,2,3,null,4,5,6], u = 4

**Output:** 5

**Explanation:** The nearest node on the same level to the right of node 4 is node 5.

My sol:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

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

res=[]

q=collections.deque()

q.append(root)

while q:

qlen=len(q)

level=[]

for x in range(qlen):

node=q.popleft()

if node:

level.append(node.val)

q.append(node.left)

q.append(node.right)

if level:

res.append(level)

print(res)

t=""

for x in res:

for y in range(0,len(x)-1):

# print(x[y],u.val)

if x[y]==u.val:

if y==len(x):

t=""

else:

t=str(x[y+1])

print(t)

if t=="":

return None

def temp(root):

if root:

print(root.val,t)

if root.val==int(t):

print("test")

self.ans=root

if root.left:

temp(root.left)

if root.right:

temp(root.right)

temp(root)

print(self.ans)

return self.ans

**1609. Even Odd Tree**

A binary tree is named **Even-Odd** if it meets the following conditions:

* The root of the binary tree is at level index 0, its children are at level index 1, their children are at level index 2, etc.
* For every **even-indexed** level, all nodes at the level have **odd** integer values in **strictly increasing** order (from left to right).
* For every **odd-indexed** level, all nodes at the level have **even** integer values in **strictly decreasing** order (from left to right).

Given the root of a binary tree, *return*true*if the binary tree is****Even-Odd****, otherwise return*false*.*

**Example 1:**

A picture containing text, sky

Description automatically generated

**Input:** root = [1,10,4,3,null,7,9,12,8,6,null,null,2]

**Output:** true

**Explanation:** The node values on each level are:

Level 0: [1]

Level 1: [10,4]

Level 2: [3,7,9]

Level 3: [12,8,6,2]

Since levels 0 and 2 are all odd and increasing and levels 1 and 3 are all even and decreasing, the tree is Even-Odd.

**My sol:**

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

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

res=[]

q=collections.deque()

q.append(root)

while q:

qlen=len(q)

level=[]

for x in range(qlen):

node=q.popleft()

if node:

level.append(node.val)

q.append(node.left)

q.append(node.right)

if level:

res.append(level)

print(res)

for x in range(0,len(res)):

#even index odd values

if x%2==0:

if len(list(set(res[x])))!=len(res[x]):

return False

if sorted(res[x])!=res[x]:

return False

for y in res[x]:

if y%2==0:

return False

#odd index even values

if x%2==1:

if len(list(set(res[x])))!=len(res[x]):

return False

if sorted(res[x])[::-1]!=res[x]:

return False

for y in res[x]:

if y%2==1:

return False

return True

**1614. Maximum Nesting Depth of the Parentheses**

A string is a **valid parentheses string** (denoted **VPS**) if it meets one of the following:

* It is an empty string "", or a single character not equal to "(" or ")",
* It can be written as AB (A concatenated with B), where A and B are **VPS**'s, or
* It can be written as (A), where A is a **VPS**.

We can similarly define the **nesting depth** depth(S) of any VPS S as follows:

* depth("") = 0
* depth(C) = 0, where C is a string with a single character not equal to "(" or ")".
* depth(A + B) = max(depth(A), depth(B)), where A and B are **VPS**'s.
* depth("(" + A + ")") = 1 + depth(A), where A is a **VPS**.

For example, "", "()()", and "()(()())" are **VPS**'s (with nesting depths 0, 1, and 2), and ")(" and "(()" are not **VPS**'s.

Given a **VPS** represented as string s, return *the****nesting depth****of*s.

**Example 1:**

**Input:** s = "(1+(2\*3)+((8)/4))+1"

**Output:** 3

**Explanation:** Digit 8 is inside of 3 nested parentheses in the string.

**My sol:**

class Solution:

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

stack=[]

count=0

maxcount=0

for x in s:

if x=='(':

stack.append(x)

count+=1

maxcount=max(count,maxcount)

elif x==')':

stack.pop()

count-=1

return maxcount

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

Given an integer array arr, return *the mean of the remaining integers after removing the smallest 5% and the largest 5% of the elements.*

Answers within 10-5 of the **actual answer** will be considered accepted.

**Example 1:**

**Input:** arr = [1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3]

**Output:** 2.00000

**Explanation:** After erasing the minimum and the maximum values of this array, all elements are equal to 2, so the mean is 2.

**My sol:**

class Solution:

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

x=0

ave=0

l=len(arr)

a=int(0.05\*l)

b=int(0.05\*l)

arr=sorted(arr)

arr=arr[a:-b]

q=sum(arr)

p=len(arr)

ave=q/p

return ave

**1624. Largest Substring Between Two Equal Characters**

Given a string s, return *the length of the longest substring between two equal characters, excluding the two characters.* If there is no such substring return -1.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** s = "aa"

**Output:** 0

**Explanation:** The optimal substring here is an empty substring between the two 'a's.

**My sol:**

class Solution:

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

# bruteforce

i=0

res=""

for x in range(0,len(s)):

while i<x:

if s[i]==s[x] and x-i+1>len(res):

res=s[i:x+1]

i+=1

i=0

if len(res)-2<-1:

return -1

else:

return len(res)-2

**1636. Sort Array by Increasing Frequency**

Given an array of integers nums, sort the array in **increasing** order based on the frequency of the values. If multiple values have the same frequency, sort them in **decreasing** order.

Return the *sorted array*.

**Example 1:**

**Input:** nums = [1,1,2,2,2,3]

**Output:** [3,1,1,2,2,2]

**Explanation:** '3' has a frequency of 1, '1' has a frequency of 2, and '2' has a frequency of 3.

**My sol:**

class Solution:

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

r = Counter(nums)

return sorted(nums, key=lambda x: (r[x], -x))

**1640. Check Array Formation Through Concatenation**

You are given an array of **distinct** integers arr and an array of integer arrays pieces, where the integers in pieces are **distinct**. Your goal is to form arr by concatenating the arrays in pieces **in any order**. However, you are **not** allowed to reorder the integers in each array pieces[i].

Return true *if it is possible to form the array*arr*from*pieces. Otherwise, return false.

**Example 1:**

**Input:** arr = [15,88], pieces = [[88],[15]]

**Output:** true

**Explanation:** Concatenate [15] then [88]

**My sol:**

class Solution:

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

d={}

for i in range(0,len(pieces)):

for y in pieces[i]:

d[y]=i

f=[]

for x in range(0,len(arr)):

if arr[x] in d and pieces[d[arr[x]]] not in f:

f.append(pieces[d[arr[x]]])

final=[]

for x in f:

for y in x:

final.append(y)

return final==arr

**1646. Get Maximum in Generated Array**

You are given an integer n. A **0-indexed** integer array nums of length n + 1 is generated in the following way:

* nums[0] = 0
* nums[1] = 1
* nums[2 \* i] = nums[i] when 2 <= 2 \* i <= n
* nums[2 \* i + 1] = nums[i] + nums[i + 1] when 2 <= 2 \* i + 1 <= n

Return*the****maximum****integer in the array*nums​​​.

**Example 1:**

**Input:** n = 7

**Output:** 3

**Explanation:** According to the given rules:

nums[0] = 0

nums[1] = 1

nums[(1 \* 2) = 2] = nums[1] = 1

nums[(1 \* 2) + 1 = 3] = nums[1] + nums[2] = 1 + 1 = 2

nums[(2 \* 2) = 4] = nums[2] = 1

nums[(2 \* 2) + 1 = 5] = nums[2] + nums[3] = 1 + 2 = 3

nums[(3 \* 2) = 6] = nums[3] = 2

nums[(3 \* 2) + 1 = 7] = nums[3] + nums[4] = 2 + 1 = 3

Hence, nums = [0,1,1,2,1,3,2,3], and the maximum is max(0,1,1,2,1,3,2,3) = 3.

**My sol:**

class Solution:

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

nums=[0,1]

if n==0:

return 0

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

if i%2==0:

nums.append(nums[i//2])

else:

nums.append(nums[i//2]+nums[i//2+1])

return max(nums)

**1647. Minimum Deletions to Make Character Frequencies Unique**

A string s is called **good** if there are no two different characters in s that have the same **frequency**.

Given a string s, return*the****minimum****number of characters you need to delete to make*s***good****.*

The **frequency** of a character in a string is the number of times it appears in the string. For example, in the string "aab", the **frequency** of 'a' is 2, while the **frequency** of 'b' is 1.

**Example 1:**

**Input:** s = "aab"

**Output:** 0

**Explanation:** s is already good.

My sol:

class Solution:

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

d=collections.Counter(s)

a=list(d.values())

if len(list(set(a)))==len(a):

return 0

count=0

for x in range(0,len(a)):

while ( a[x] in a[x+1:] or a[x] in a[:x] )and a[x]!=0:

a[x]-=1

count+=1

return count

1651-1700 ($$$$$ 12)

**1662. Check If Two String Arrays are Equivalent**

Given two string arrays word1 and word2, returntrue*if the two arrays****represent****the same string, and*false*otherwise.*

A string is **represented** by an array if the array elements concatenated **in order** forms the string.

**Example 1:**

**Input:** word1 = ["ab", "c"], word2 = ["a", "bc"]

**Output:** true

**Explanation:**

word1 represents string "ab" + "c" -> "abc"

word2 represents string "a" + "bc" -> "abc"

The strings are the same, so return true.

**My sol:**

class Solution:

def arrayStringsAreEqual(self, word1: List[str], word2: List[str]) -> bool:

s1=''

s3=s1.join(word1)

s2=''

s4=s2.join(word2)

return s3 == s4

**1668. Maximum Repeating Substring**For a string sequence, a string word is **k-repeating** if word concatenated k times is a substring of sequence. The word's **maximum k-repeating value** is the highest value k where word is k-repeating in sequence. If word is not a substring of sequence, word's maximum k-repeating value is 0.

Given strings sequence and word, return *the****maximum k-repeating value****of word in sequence*.

**Example 1:**

**Input:** sequence = "ababc", word = "ab"

**Output:** 2

**Explanation:** "abab" is a substring in "ababc".

**My sol:**

class Solution:

def maxRepeating(self, sequence: str, word: str) -> int:

count=0

a=word

while word in sequence:

word+=a

count+=1

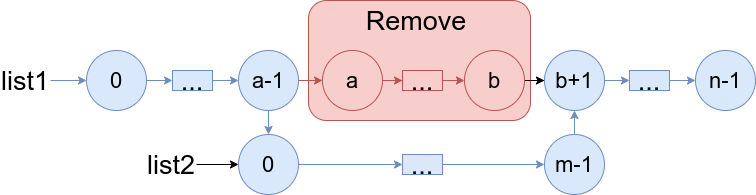
return count

**1669. Merge In Between Linked Lists**

You are given two linked lists: list1 and list2 of sizes n and m respectively.

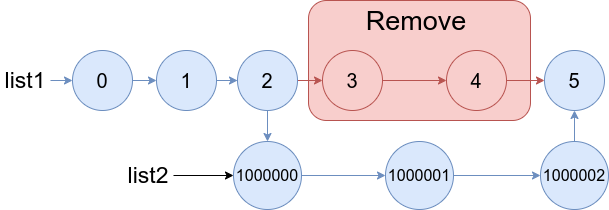
Remove list1's nodes from the ath node to the bth node, and put list2 in their place.

The blue edges and nodes in the following figure indicate the result:



*Build the result list and return its head.*

**Example 1:**



**Input:** list1 = [0,1,2,3,4,5], a = 3, b = 4, list2 = [1000000,1000001,1000002]

**Output:** [0,1,2,1000000,1000001,1000002,5]

**Explanation:** We remove the nodes 3 and 4 and put the entire list2 in their place. The blue edges and nodes in the above figure indicate the result.

**My sol:**

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def mergeInBetween(self, list1: ListNode, a: int, b: int, list2: ListNode) -> ListNode:

dummy=ListNode()

p=dummy

cur1=list1

s1=[]

while cur1:

s1.append(cur1.val)

cur1=cur1.next

cur2=list2

s2=[]

while cur2:

s2.append(cur2.val)

cur2=cur2.next

temp=s1[:a]+s2+s1[b+1:]

for x in temp:

p.next=ListNode(x)

p=p.next

return dummy.next

**1672. Richest Customer Wealth**

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the i​​​​​​​​​​​th​​​​ customer has in the j​​​​​​​​​​​th​​​​ bank. Return*the****wealth****that the richest customer has.*

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

**Example 1:**

**Input:** accounts = [[1,2,3],[3,2,1]]

**Output:** 6

**Explanation:**

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

My sol:

class Solution:

def maximumWealth(self, accounts: List[List[int]]) -> int:

totalw=[]

temp=0

for i in range(0,len(accounts)):

for y in accounts[i]:

temp =temp +y

totalw.append(temp)

temp=0

return(max(totalw))

**1678. Goal Parser Interpretation**

You own a **Goal Parser** that can interpret a string command. The command consists of an alphabet of "G", "()" and/or "(al)" in some order. The Goal Parser will interpret "G" as the string "G", "()" as the string "o", and "(al)" as the string "al". The interpreted strings are then concatenated in the original order.

Given the string command, return *the****Goal Parser****'s interpretation of*command.

**Example 1:**

**Input:** command = "G()(al)"

**Output:** "Goal"

**Explanation:** The Goal Parser interprets the command as follows:

G -> G

() -> o

(al) -> al

The final concatenated result is "Goal".

My sol:

class Solution:

def interpret(self, command: str) -> str:

if '()' in command:

command=command.replace('()','o')

if '(al)' in command:

command=command.replace('(al)','al')

return command

**1680. Concatenation of Consecutive Binary Numbers**

Given an integer n, return *the****decimal value****of the binary string formed by concatenating the binary representations of*1*to*n*in order,****modulo***109+ 7.

**Example 1:**

**Input:** n = 1

**Output:** 1

**Explanation:** "1" in binary corresponds to the decimal value 1.

My sol:

class Solution:

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

s=""

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

s+=bin(x)[2:]

return (int(s,2)%(10\*\*9+7))

**1684. Count the Number of Consistent Strings**

You are given a string allowed consisting of **distinct** characters and an array of strings words. A string is **consistent**if all characters in the string appear in the string allowed.

Return*the number of****consistent****strings in the array*words.

**Example 1:**

**Input:** allowed = "ab", words = ["ad","bd","aaab","baa","badab"]

**Output:** 2

**Explanation:** Strings "aaab" and "baa" are consistent since they only contain characters 'a' and 'b'.

My sol:

class Solution:

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

count=0

flag=0

for x in words:

for y in x:

if y in allowed:

flag=0

else:

flag=1

break

if flag==0:

count=count+1

else:

pass

flag=0

return count

**1688. Count of Matches in Tournament**

You are given an integer n, the number of teams in a tournament that has strange rules:

* If the current number of teams is **even**, each team gets paired with another team. A total of n / 2 matches are played, and n / 2 teams advance to the next round.
* If the current number of teams is **odd**, one team randomly advances in the tournament, and the rest gets paired. A total of (n - 1) / 2 matches are played, and (n - 1) / 2 + 1 teams advance to the next round.

Return *the number of matches played in the tournament until a winner is decided.*

**Example 1:**

**Input:** n = 7

**Output:** 6

**Explanation:** Details of the tournament:

- 1st Round: Teams = 7, Matches = 3, and 4 teams advance.

- 2nd Round: Teams = 4, Matches = 2, and 2 teams advance.

- 3rd Round: Teams = 2, Matches = 1, and 1 team is declared the winner.

Total number of matches = 3 + 2 + 1 = 6.

My sol:

class Solution:

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

matchesplayed=0

while n!=1:

if n%2==0 :

matchesplayed+=n-(n//2)

n=n//2

else:

matchesplayed+=n-((n-1)//2)

n=((n-1)//2)

return matchesplayed

**1689. Partitioning Into Minimum Number Of Deci-Binary Numbers**

A decimal number is called **deci-binary** if each of its digits is either 0 or 1 without any leading zeros. For example, 101 and 1100 are **deci-binary**, while 112 and 3001 are not.

Given a string n that represents a positive decimal integer, return *the****minimum****number of positive****deci-binary****numbers needed so that they sum up to*n*.*

**Example 1:**

**Input:** n = "32"

**Output:** 3

**Explanation:** 10 + 11 + 11 = 32

My sol:

class Solution:

def minPartitions(self, n: str) -> int:

m=-1

for x in str(n):

if int(x) >m:

m=int(x)

return m

**1694. Reformat Phone Number**

You are given a phone number as a string number. number consists of digits, spaces ' ', and/or dashes '-'.

You would like to reformat the phone number in a certain manner. Firstly, **remove** all spaces and dashes. Then, **group** the digits from left to right into blocks of length 3 **until** there are 4 or fewer digits. The final digits are then grouped as follows:

* 2 digits: A single block of length 2.
* 3 digits: A single block of length 3.
* 4 digits: Two blocks of length 2 each.

The blocks are then joined by dashes. Notice that the reformatting process should **never** produce any blocks of length 1 and produce **at most** two blocks of length 2.

Return *the phone number after formatting.*

**Example 1:**

**Input:** number = "1-23-45 6"

**Output:** "123-456"

**Explanation:** The digits are "123456".

Step 1: There are more than 4 digits, so group the next 3 digits. The 1st block is "123".

Step 2: There are 3 digits remaining, so put them in a single block of length 3. The 2nd block is "456".

Joining the blocks gives "123-456".

**My sol:**

class Solution:

def reformatNumber(self, number: str) -> str:

if len(str(number))<=3:

return str(number)

a=str(number)

a=a.replace('-','')

a=a.replace(' ','')

l=len(a)%3

i=3

while i<len(a) and len(a[i:])>2:

a=a[:i]+'-'+a[i:]

i+=4

if l%3==0:

pass

if l%3==1:

a=a[:l-3]+"-"+a[l-3:]

if l%3==2:

a=a[:l-4]+"-"+a[l-4:]

return a

**1698. Number of Distinct Substrings in a String**

Given a string s, return *the number of****distinct****substrings of* s.

A **substring** of a string is obtained by deleting any number of characters (possibly zero) from the front of the string and any number (possibly zero) from the back of the string.

**Example1:**

**Input:** s = "aabbaba"

**Output:** 21

**Explanation:** The set of distinct strings is ["a","b","aa","bb","ab","ba","aab","abb","bab","bba","aba","aabb","abba","bbab","baba","aabba","abbab","bbaba","aabbab","abbaba","aabbaba"]

My sol:

class Solution:

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

i=0

j=0

a=set()

count=0

while i<=len(s):

while j<=i:

if s[j:i] in a:

pass

else:

a.add(s[j:i])

count+=1

j+=1

i+=1

j=0

return count-1

**1700. Number of Students Unable to Eat Lunch**

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a **stack**. At each step:

* If the student at the front of the queue **prefers** the sandwich on the top of the stack, they will **take it** and leave the queue.
* Otherwise, they will **leave it** and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the i​​​​​​th sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the j​​​​​​th student in the initial queue (j = 0 is the front of the queue). Return *the number of students that are unable to eat.*

**Example 1:**

**Input:** students = [1,1,0,0], sandwiches = [0,1,0,1]

**Output:** 0

**Explanation:**

- Front student leaves the top sandwich and returns to the end of the line making students = [1,0,0,1].

- Front student leaves the top sandwich and returns to the end of the line making students = [0,0,1,1].

- Front student takes the top sandwich and leaves the line making students = [0,1,1] and sandwiches = [1,0,1].

- Front student leaves the top sandwich and returns to the end of the line making students = [1,1,0].

- Front student takes the top sandwich and leaves the line making students = [1,0] and sandwiches = [0,1].

- Front student leaves the top sandwich and returns to the end of the line making students = [0,1].

- Front student takes the top sandwich and leaves the line making students = [1] and sandwiches = [1].

- Front student takes the top sandwich and leaves the line making students = [] and sandwiches = [].

Hence all students are able to eat.

**My sol:**

class Solution:

def countStudents(self, students: List[int], sandwiches: List[int]) -> int:

a=[]

while students:

if students and students[0]==sandwiches[0]:

sandwiches.pop(0)

students.pop(0)

else:

a=students.pop(0)

students.append(a)

if sandwiches and sandwiches[0] not in students:

return len(students)

if students==sandwiches:

return len(students)

1701-1750 ($$$$$ 12)

**1701. Average Waiting Time**

There is a restaurant with a single chef. You are given an array customers, where customers[i] = [arrivali, timei]:

* arrivali is the arrival time of the ith customer. The arrival times are sorted in **non-decreasing** order.
* timei is the time needed to prepare the order of the ith customer.

When a customer arrives, he gives the chef his order, and the chef starts preparing it once he is idle. The customer waits till the chef finishes preparing his order. The chef does not prepare food for more than one customer at a time. The chef prepares food for customers **in the order they were given in the input**.

Return *the****average****waiting time of all customers*. Solutions within 10-5 from the actual answer are considered accepted.

**Example 1:**

**Input:** customers = [[1,2],[2,5],[4,3]]

**Output:** 5.00000

**Explanation:**

1) The first customer arrives at time 1, the chef takes his order and starts preparing it immediately at time 1, and finishes at time 3, so the waiting time of the first customer is 3 - 1 = 2.

2) The second customer arrives at time 2, the chef takes his order and starts preparing it at time 3, and finishes at time 8, so the waiting time of the second customer is 8 - 2 = 6.

3) The third customer arrives at time 4, the chef takes his order and starts preparing it at time 8, and finishes at time 11, so the waiting time of the third customer is 11 - 4 = 7.

**My soL:**

class Solution:

def averageWaitingTime(self, customers: List[List[int]]) -> float:

v=0

temp=0

count=0

for x in customers:

v=max(x[0],temp)

temp=v+x[1]

count+=temp-x[0]

return count/len(customers)

**1704. Determine if String Halves Are Alike**

You are given a string s of even length. Split this string into two halves of equal lengths, and let a be the first half and b be the second half.

Two strings are **alike** if they have the same number of vowels ('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'). Notice that s contains uppercase and lowercase letters.

Return true*if*a*and*b*are****alike***. Otherwise, return false.

**Example 1:**

**Input:** s = "book"

**Output:** true

**Explanation:** a = "bo" and b = "ok". a has 1 vowel and b has 1 vowel. Therefore, they are alike.

**My sol:**

class Solution:

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

count1=count2=0

ovels=['a','e','i','o','u','A','E','I','O','U']

for i in s[0:int(len(s)/2)]:

if i in ovels:

count1+=1

for i in s[int(len(s)/2):]:

if i in ovels:

count2+=1

return count1==count2

**1708. Largest Subarray Length K**

An array A is larger than some array B if for the first index i where A[i] != B[i], A[i] > B[i].

For example, consider 0-indexing:

* [1,3,2,4] > [1,2,2,4], since at index 1, 3 > 2.
* [1,4,4,4] < [2,1,1,1], since at index 0, 1 < 2.

A subarray is a contiguous subsequence of the array.

Given an integer array nums of **distinct** integers, return the **largest** subarray of nums of length k.

**Example 1:**

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

**Output:** [5,2,3]

**Explanation:** The subarrays of size 3 are: [1,4,5], [4,5,2], and [5,2,3].

Of these, [5,2,3] is the largest.

**My sol:**

class Solution:

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

a=[]

if nums.index(max(nums))+k<=len(nums):

return nums[nums.index(max(nums)):nums.index(max(nums))+k]

for i in range(0,len(nums)):

if i+k<=len(nums):

if nums[i:i+k]>a:

a=nums[i:i+k]

return a

**1710. Maximum Units on a Truck**

**(2d array reverse sort based on one attribute using lambda)**

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]:

* numberOfBoxesi is the number of boxes of type i.
* numberOfUnitsPerBoxiis the number of units in each box of the type i.

You are also given an integer truckSize, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize.

Return *the****maximum****total number of****units****that can be put on the truck.*

**Example 1:**

**Input:** boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4

**Output:** 8

**Explanation:** There are:

- 1 box of the first type that contains 3 units.

- 2 boxes of the second type that contain 2 units each.

- 3 boxes of the third type that contain 1 unit each.

You can take all the boxes of the first and second types, and one box of the third type.

The total number of units will be = (1 \* 3) + (2 \* 2) + (1 \* 1) = 8.

**My sol:**

class Solution:

def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:

temp = sorted(boxTypes, key=lambda x: x[1], reverse=True)

print(temp)

res=0

for x in temp:

if x[0]<=truckSize:

res+=x[0]\*x[1]

elif truckSize<x[0]:

res+=truckSize\*x[1]

break

truckSize-=x[0]

return res

**1716. Calculate Money in Leetcode Bank**

Hercy wants to save money for his first car. He puts money in the Leetcode bank **every day**.

He starts by putting in $1 on Monday, the first day. Every day from Tuesday to Sunday, he will put in $1 more than the day before. On every subsequent Monday, he will put in $1 more than the **previous Monday**.

Given n, return *the total amount of money he will have in the Leetcode bank at the end of the*nth*day.*

**Example 1:**

**Input:** n = 4

**Output:** 10

**Explanation:** After the 4th day, the total is 1 + 2 + 3 + 4 = 10.

**My sol:**

class Solution:

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

a=[0 for x in range(n)]

a[0]=1

for x in range(1,n):

if x%7==0:

a[x]=a[x-7]+1

else:

a[x]=a[x-1]+1

print(a)

return sum(a[:n])

**1720. Decode XORed Array**

There is a **hidden** integer array arr that consists of n non-negative integers.

It was encoded into another integer array encoded of length n - 1, such that encoded[i] = arr[i] XOR arr[i + 1]. For example, if arr = [1,0,2,1], then encoded = [1,2,3].

You are given the encoded array. You are also given an integer first, that is the first element of arr, i.e. arr[0].

Return *the original array* arr. It can be proved that the answer exists and is unique.

**Example 1:**

**Input:** encoded = [1,2,3], first = 1

**Output:** [1,0,2,1]

**Explanation:** If arr = [1,0,2,1], then first = 1 and encoded = [1 XOR 0, 0 XOR 2, 2 XOR 1] = [1,2,3]

**My sol:**

class Solution:

def decode(self, encoded: List[int], first: int) -> List[int]:

a=[first]

for x in encoded:

b=first^x

first=b

a.append(b)

return a

**1721. Swapping Nodes in a Linked List**

You are given the head of a linked list, and an integer k.

Return *the head of the linked list after****swapping****the values of the*kth *node from the beginning and the*kth *node from the end (the list is****1-indexed****).*

**Example 1:**

Diagram

Description automatically generated

**Input:** head = [1,2,3,4,5], k = 2

**Output:** [1,4,3,2,5]

My sol:

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def swapNodes(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:

dummy=ListNode()

p=dummy

cur=head

temp=[]

while cur:

temp.append(cur.val)

cur=cur.next

a=temp[k-1]

b=temp[len(temp)-k]

temp[k-1]=b

temp[len(temp)-k]=a

a=temp

for x in a:

p.next=ListNode(x)

p=p.next

return dummy.next

**1725. Number Of Rectangles That Can Form The Largest Square**

You are given an array rectangles where rectangles[i] = [li, wi] represents the ith rectangle of length li and width wi.

You can cut the ith rectangle to form a square with a side length of k if both k <= li and k <= wi. For example, if you have a rectangle [4,6], you can cut it to get a square with a side length of at most 4.

Let maxLen be the side length of the **largest** square you can obtain from any of the given rectangles.

Return *the****number****of rectangles that can make a square with a side length of*maxLen.

**Example 1:**

**Input:** rectangles = [[5,8],[3,9],[5,12],[16,5]]

**Output:** 3

**Explanation:** The largest squares you can get from each rectangle are of lengths [5,3,5,5].

The largest possible square is of length 5, and you can get it out of 3 rectangles.

**My sol:**

class Solution:

def countGoodRectangles(self, rectangles: List[List[int]]) -> int:

a=[]

for x in rectangles:

a.append(min(x[0],x[1]))

t=max(a)

return a.count(t)

**1732. Find the Highest Altitude**

There is a biker going on a road trip. The road trip consists of n + 1 points at different altitudes. The biker starts his trip on point 0 with altitude equal 0.

You are given an integer array gain of length n where gain[i] is the **net gain in altitude** between points i​​​​​​ and i + 1 for all (0 <= i < n). Return *the****highest altitude****of a point.*

**Example 1:**

**Input:** gain = [-5,1,5,0,-7]

**Output:** 1

**Explanation:** The altitudes are [0,-5,-4,1,1,-6]. The highest is 1.

**My sol:**

class Solution:

def largestAltitude(self, gain: List[int]) -> int:

alt=[0]

for i in range(0,len(gain)):

s=alt[i]+gain[i]

alt.append(s)

print(alt)

return max(alt)

**1742. Maximum Number of Balls in a Box**

You are working in a ball factory where you have n balls numbered from lowLimit up to highLimit **inclusive** (i.e., n == highLimit - lowLimit + 1), and an infinite number of boxes numbered from 1 to infinity.

Your job at this factory is to put each ball in the box with a number equal to the sum of digits of the ball's number. For example, the ball number 321 will be put in the box number 3 + 2 + 1 = 6 and the ball number 10 will be put in the box number 1 + 0 = 1.

Given two integers lowLimit and highLimit, return*the number of balls in the box with the most balls.*

**Example 1:**

**Input:** lowLimit = 1, highLimit = 10

**Output:** 2

**Explanation:**

Box Number: 1 2 3 4 5 6 7 8 9 10 11 ...

Ball Count: 2 1 1 1 1 1 1 1 1 0 0 ...

Box 1 has the most number of balls with 2 balls.

**My sol:**

class Solution:

def countBalls(self, lowLimit: int, highLimit: int) -> int:

boxes = [0] \* 100

for i in range(lowLimit, highLimit + 1):

boxes[sum([int(j) for j in str(i)])] += 1

return max(boxes)

**1748. Sum of Unique Elements**

You are given an integer array nums. The unique elements of an array are the elements that appear **exactly once** in the array.

Return *the****sum****of all the unique elements of*nums.

**Example 1:**

**Input:** nums = [1,2,3,2]

**Output:** 4

**Explanation:** The unique elements are [1,3], and the sum is 4.

**My sol:**

class Solution:

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

final=[]

temp=[]

for i in nums:

if i not in final and i not in temp:

final.append(i)

elif i not in final and i in temp:

pass

elif i in final:

final.remove(i)

temp.append(i)

if len(nums)!=1 and len(final)==1:

final=[]

print(final)

b= sum(final)

return b

**more optimized:**

count=0

d=collections.Counter(nums)

for k,v in d.items():

if v<=1:

count+=k

return count

**1749. Maximum Absolute Sum of Any Subarray**

**(subarray logic with flag like 53 and 2 flag resets )**

You are given an integer array nums. The **absolute sum** of a subarray [numsl, numsl+1, ..., numsr-1, numsr] is abs(numsl + numsl+1 + ... + numsr-1 + numsr).

Return *the****maximum****absolute sum of any****(possibly empty)****subarray of*nums.

Note that abs(x) is defined as follows:

* If x is a negative integer, then abs(x) = -x.
* If x is a non-negative integer, then abs(x) = x.

**Example 1:**

**Input:** nums = [1,-3,2,3,-4]

**Output:** 5

**Explanation:** The subarray [2,3] has absolute sum = abs(2+3) = abs(5) = 5.

**My sol:**

class Solution:

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

lsum=nums[0]

tsum=0

nsum=0

for x in nums:

if tsum<0:

tsum=0

if nsum>0:

nsum=0

tsum+=x

nsum+=x

lsum=max(tsum,lsum,-nsum)

return lsum

1751-1800 ($$$$$ 15)

**1752. Check if Array Is Sorted and Rotated**

Given an array nums, return true*if the array was originally sorted in non-decreasing order, then rotated****some****number of positions (including zero)*. Otherwise, return false.

There may be **duplicates** in the original array.

**Note:** An array A rotated by x positions results in an array B of the same length such that A[i] == B[(i+x) % A.length], where % is the modulo operation.

**Example 1:**

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

**Output:** true

**Explanation:** [1,2,3,4,5] is the original sorted array.

You can rotate the array by x = 3 positions to begin on the the element of value 3: [3,4,5,1,2].

My sol:

class Solution:

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

for i in range(0,len(nums)):

if nums[i:] + nums[:i] == sorted (nums):

return True

**1758. Minimum Changes To Make Alternating Binary String**

You are given a string s consisting only of the characters '0' and '1'. In one operation, you can change any '0' to '1' or vice versa.

The string is called alternating if no two adjacent characters are equal. For example, the string "010" is alternating, while the string "0100" is not.

Return *the****minimum****number of operations needed to make* s *alternating*.

**Example 1:**

**Input:** s = "0100"

**Output:** 1

**Explanation:** If you change the last character to '1', s will be "0101", which is alternating.

**My sol:**

class Solution:

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

temp1='0'

i=1

temp2='1'

j=1

while i<len(s):

if i % 2==0:

temp1+='0'

else:

temp1+='1'

i+=1

print(temp1)

while j<len(s):

if j % 2==0:

temp2+='1'

else:

temp2+='0'

j+=1

print(temp2)

count1=0

count2=0

for x in range(0,len(s)):

if s[x]!=temp1[x]:

count1+=1

for x in range(0,len(s)):

if s[x]!=temp2[x]:

count2+=1

return min(count1,count2)

**1759. Count Number of Homogenous Substrings O(N)**

Given a string s, return *the number of****homogenous****substrings of*s*.* Since the answer may be too large, return it **modulo** 109 + 7.

A string is **homogenous** if all the characters of the string are the same.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** s = "abbcccaa"

**Output:** 13

**Explanation:** The homogenous substrings are listed as below:

"a" appears 3 times.

"aa" appears 1 time.

"b" appears 2 times.

"bb" appears 1 time.

"c" appears 3 times.

"cc" appears 2 times.

"ccc" appears 1 time.

3 + 1 + 2 + 1 + 3 + 2 + 1 = 13.

My sol:

class Solution:

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

res=0

temp=0

var=s[0]

d={}

for x in range(0,len(s)):

if s[x] in d and var==s[x]:

d[s[x]]+=1

else:

d={}

d[s[x]]=1

var=s[x]

temp+=d[s[x]]

res+=temp

temp=0

return res % ((10\*\*9)+7)

**1762. Buildings With an Ocean View**

There are n buildings in a line. You are given an integer array heights of size n that represents the heights of the buildings in the line.

The ocean is to the right of the buildings. A building has an ocean view if the building can see the ocean without obstructions. Formally, a building has an ocean view if all the buildings to its right have a **smaller** height.

Return a list of indices **(0-indexed)** of buildings that have an ocean view, sorted in increasing order.

**Example 1:**

**Input:** heights = [4,2,3,1]

**Output:** [0,2,3]

**Explanation:** Building 1 (0-indexed) does not have an ocean view because building 2 is taller.

My sol:

class Solution:

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

res=[]

maxheight=0

for x in range(len(heights)-1,-1,-1):

if heights[x]>maxheight:

res.append(x)

maxheight=heights[x]

print(res)

return res[::-1]

**1763. Longest Nice Substring**

A string s is **nice** if, for every letter of the alphabet that s contains, it appears **both** in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

Given a string s, return *the longest****substring****of s that is****nice****. If there are multiple, return the substring of the****earliest****occurrence. If there are none, return an empty string*.

**Example 1:**

**Input:** s = "YazaAay"

**Output:** "aAa"

**Explanation:** "aAa" is a nice string because 'A/a' is the only letter of the alphabet in s, and both 'A' and 'a' appear.

"aAa" is the longest nice substring.

**My sol:**

class Solution:

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

i=0

j=0

flag=0

res=""

while i<=len(s):

j=0

while j<i:

flag=0

for p in s[j:i]:

if p.upper() not in s[j:i] or p.lower() not in s[j:i]:

flag=1

break

if flag==0 and len(s[j:i])!=0:

if len(s[j:i])>len(res):

res=s[j:i]

j+=1

i+=1

return res

**1768. Merge Strings Alternately**

You are given two strings word1 and word2. Merge the strings by adding letters in alternating order, starting with word1. If a string is longer than the other, append the additional letters onto the end of the merged string.

Return *the merged string.*

**Example 1:**

**Input:** word1 = "abc", word2 = "pqr"

**Output:** "apbqcr"

**Explanation:** The merged string will be merged as so:

word1: a b c

word2: p q r

merged: a p b q c r

My sol:

class Solution:

def mergeAlternately(self, word1: str, word2: str) -> str:

o=''

l=min(len(word1),len(word2))

for i in range(0,l):

o= o + word1[i]

o= o+ word2[i]

if len(word1)>len(word2):

o= o + str(word1[l:])

elif len(word2)>len(word1):

o= o + str(word2[l:])

return o

**1769. Minimum Number of Operations to Move All Balls to Each Box**

You have n boxes. You are given a binary string boxes of length n, where boxes[i] is '0' if the ith box is **empty**, and '1' if it contains **one** ball.

In one operation, you can move **one** ball from a box to an adjacent box. Box i is adjacent to box j if abs(i - j) == 1. Note that after doing so, there may be more than one ball in some boxes.

Return an array answer of size n, where answer[i] is the **minimum** number of operations needed to move all the balls to the ith box.

Each answer[i] is calculated considering the **initial** state of the boxes.

**Example 1:**

**Input:** boxes = "110"

**Output:** [1,1,3]

**Explanation:** The answer for each box is as follows:

1) First box: you will have to move one ball from the second box to the first box in one operation.

2) Second box: you will have to move one ball from the first box to the second box in one operation.

3) Third box: you will have to move one ball from the first box to the third box in two operations, and move one ball from the second box to the third box in one operation.

My sol:

class Solution:

def minOperations(self, boxes: str) -> List[int]:

dict={}

for x in range(0,len(boxes)):

dict[x]=boxes[x]

sum=0

temp=[]

for k,v in dict.items():

if v=='1':

temp.append(k)

t=0

final=[]

for x in range(0,len(boxes)):

for y in temp:

t+=abs(y-x)

final.append(t)

t=0

return final

**1773. Count Items Matching a Rule**

You are given an array items, where each items[i] = [typei, colori, namei] describes the type, color, and name of the ith item. You are also given a rule represented by two strings, ruleKey and ruleValue.

The ith item is said to match the rule if **one** of the following is true:

* ruleKey == "type" and ruleValue == typei.
* ruleKey == "color" and ruleValue == colori.
* ruleKey == "name" and ruleValue == namei.

Return *the number of items that match the given rule*.

**Example 1:**

**Input:** items = [["phone","blue","pixel"],["computer","silver","lenovo"],["phone","gold","iphone"]], ruleKey = "color", ruleValue = "silver"

**Output:** 1

**Explanation:** There is only one item matching the given rule, which is ["computer","silver","lenovo"].

My sol:

class Solution:

def countMatches(self, items: List[List[str]], ruleKey: str, ruleValue: str) -> int:

count=[]

for i in range(0,len(items)):

if ruleKey=="type":

count.append(items[i][0]==ruleValue)

elif ruleKey=="color":

count.append(items[i][1]==ruleValue)

elif ruleKey=="name":

count.append(items[i][2]==ruleValue)

return sum(count)

**1779. Find Nearest Point That Has the Same X or Y Coordinate**

You are given two integers, x and y, which represent your current location on a Cartesian grid: (x, y). You are also given an array points where each points[i] = [ai, bi] represents that a point exists at (ai, bi). A point is **valid** if it shares the same x-coordinate or the same y-coordinate as your location.

Return *the index****(0-indexed)****of the****valid****point with the smallest****Manhattan distance****from your current location*. If there are multiple, return *the valid point with the****smallest****index*. If there are no valid points, return -1.

The **Manhattan distance** between two points (x1, y1) and (x2, y2) is abs(x1 - x2) + abs(y1 - y2).

**Example 1:**

**Input:** x = 3, y = 4, points = [[1,2],[3,1],[2,4],[2,3],[4,4]]

**Output:** 2

**Explanation:** Of all the points, only [3,1], [2,4] and [4,4] are valid. Of the valid points, [2,4] and [4,4] have the smallest Manhattan distance from your current location, with a distance of 1. [2,4] has the smallest index, so return 2.

**My sol:**

class Solution:

def nearestValidPoint(self, x: int, y: int, points: List[List[int]]) -> int:

dummy=[]

ans=[]

for a in points:

if a[0]==x or a[1]==y:

dummy.append(a)

else:

pass

for a in dummy:

ans.append(abs(x-a[0])+abs(y-a[1]))

if len(dummy)==0:

return -1

else:

for a in dummy:

if abs(x-a[0])+abs(y-a[1]) == int(min(ans)):

return points.index(a)

**1784. Check if Binary String Has at Most One Segment of Ones**

Given a binary string s **​​​​​without leading zeros**, return true​​​ *if*s*contains****at most one contiguous segment of ones***. Otherwise, return false.

**Example 1:**

**Input:** s = "1001"

**Output:** false

**Explanation:** The ones do not form a contiguous segment.

**My sol:**

class Solution:

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

flag=0

count=0

for x in range(0,len(s)-1):

if s[x]=='1' and s[x+1]!='1':

count+=1

if s[x]!='1' and s[x+1]=='1':

count+=1

print(count)

return count<=1

**1784. Check if Binary String Has at Most One Segment of Ones**

Given a binary string s **​​​​​without leading zeros**, return true​​​ *if*s*contains****at most one contiguous segment of ones***. Otherwise, return false.

**Example 1:**

**Input:** s = "1001"

**Output:** false

**Explanation:** The ones do not form a contiguous segment.

My sol:

class Solution:

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

flag=0

count=0

for x in range(0,len(s)-1):

if s[x]=='1' and s[x+1]!='1':

count+=1

if s[x]!='1' and s[x+1]=='1':

count+=1

print(count)

return count<=1

**1790. Check if One String Swap Can Make Strings Equal**

You are given two strings s1 and s2 of equal length. A **string swap** is an operation where you choose two indices in a string (not necessarily different) and swap the characters at these indices.

Return true *if it is possible to make both strings equal by performing****at most one string swap****on****exactly one****of the strings.*Otherwise, return false.

**Example 1:**

**Input:** s1 = "bank", s2 = "kanb"

**Output:** true

**Explanation:** For example, swap the first character with the last character of s2 to make "bank".

**My sol:**

class Solution:

def areAlmostEqual(self, s1: str, s2: str) -> bool:

count=0

b=[]

for x in range(0,len(s1)):

if s1[x]!=s2[x]:

count+=1

b.append(x)

if count>2:

return False

b.sort()

if len(b)==0:

return True

if len(b)==1 and s1[b[0]]!=s2[b[0]]:

return False

if s1[b[0]]!=s2[b[1]] or s2[b[0]]!=s1[b[1]]:

return False

return True

**1791. Find Center of Star Graph**

There is an undirected **star** graph consisting of n nodes labeled from 1 to n. A star graph is a graph where there is one **center** node and **exactly** n - 1 edges that connect the center node with every other node.

You are given a 2D integer array edges where each edges[i] = [ui, vi] indicates that there is an edge between the nodes ui and vi. Return the center of the given star graph.

**Example 1:**

A picture containing text, pool ball, vector graphics

Description automatically generated

**Input:** edges = [[1,2],[2,3],[4,2]]

**Output:** 2

**Explanation:** As shown in the figure above, node 2 is connected to every other node, so 2 is the center.

**My sol:**

class Solution:

def findCenter(self, edges: List[List[int]]) -> int:

[a,b]=edges[0]

flag=0

final=0

for x in edges:

if a in x:

flag=0

final=a

else:

flag=1

break

if flag==0:

return final

return b

**1796. Second Largest Digit in a String**

Given an alphanumeric string s, return *the****second largest****numerical digit that appears in*s*, or*-1*if it does not exist*.

An **alphanumeric**string is a string consisting of lowercase English letters and digits.

**Example 1:**

**Input:** s = "dfa12321afd"

**Output:** 2

**Explanation:** The digits that appear in s are [1, 2, 3]. The second largest digit is 2.

My soL:

class Solution:

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

p=[]

for x in s:

if 48<= ord(x) <=57:

# print(x)

p.append(x)

print(p)

if not p:

pass

else:

m1=max(p)

try:

while True:

p.remove(m1)

except ValueError:

pass

if not p:

return -1

else:

return max(p)

**1800. Maximum Ascending Subarray Sum (subarray with flag and O(N)**

Given an array of positive integers nums, return the *maximum possible sum of an****ascending****subarray in*nums.

A subarray is defined as a contiguous sequence of numbers in an array.

A subarray [numsl, numsl+1, ..., numsr-1, numsr] is **ascending** if for all i where l <= i < r, numsi< numsi+1. Note that a subarray of size 1 is **ascending**.

**Example 1:**

**Input:** nums = [10,20,30,5,10,50]

**Output:** 65

**Explanation:** [5,10,50] is the ascending subarray with the maximum sum of 65.

**My sol:**

class Solution:

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

temp=nums[0]

res=temp

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

if nums[x]<=nums[x-1]:

temp=0

temp=temp+nums[x]

res=max(temp,res)

return res

1801-1850 ($$$$$ 13)

**1805. Number of Different Integers in a String ( number regex)**

You are given a string word that consists of digits and lowercase English letters.

You will replace every non-digit character with a space. For example, "a123bc34d8ef34" will become " 123  34 8  34". Notice that you are left with some integers that are separated by at least one space: "123", "34", "8", and "34".

Return *the number of****different****integers after performing the replacement operations on*word.

Two integers are considered different if their decimal representations **without any leading zeros** are different.

**Example 1:**

**Input:** word = "a123bc34d8ef34"

**Output:** 3

**Explanation:** The three different integers are "123", "34", and "8". Notice that "34" is only counted once.

**My soL:**

class Solution:

def numDifferentIntegers(self, word: str) -> int:

word = re.findall('(\d+)', word)

numbers = [int(i) for i in word]

print(numbers)

return len(set(numbers))

**1812. Determine Color of a Chessboard Square**

You are given coordinates, a string that represents the coordinates of a square of the chessboard. Below is a chessboard for your reference.

A black and white checkered surface

Description automatically generated with medium confidence

Return true*if the square is white, and*false*if the square is black*.

The coordinate will always represent a valid chessboard square. The coordinate will always have the letter first, and the number second.

**Example 1:**

**Input:** coordinates = "a1"

**Output:** false

**Explanation:** From the chessboard above, the square with coordinates "a1" is black, so return false.

**My sol:**

class Solution:

def squareIsWhite(self, coordinates: str) -> bool:

if coordinates=='a1':

return False

elif coordinates=='a2':

return True

elif coordinates=='a3':

return False

elif coordinates=='a4':

return True

elif coordinates=='a5':

return False

elif coordinates=='a6':

return True

elif coordinates=='a7':

return False

elif coordinates=='a8':

return True

elif coordinates=='b1':

return True

elif coordinates=='b2':

return False

elif coordinates=='b3':

return True

elif coordinates=='b4':

return False

elif coordinates=='b5':

return True

elif coordinates=='b6':

return False

elif coordinates=='b7':

return True

elif coordinates=='b8':

return False

elif coordinates=='c1':

return False

elif coordinates=='c2':

return True

elif coordinates=='c3':

return False

elif coordinates=='c4':

return True

elif coordinates=='c5':

return False

elif coordinates=='c6':

return True

elif coordinates=='c7':

return False

elif coordinates=='c8':

return True

elif coordinates=='d1':

return True

elif coordinates=='d2':

return False

elif coordinates=='d3':

return True

elif coordinates=='d4':

return False

elif coordinates=='d5':

return True

elif coordinates=='d6':

return False

elif coordinates=='d7':

return True

elif coordinates=='d8':

return False

elif coordinates=='e1':

return False

elif coordinates=='e2':

return True

elif coordinates=='e3':

return False

elif coordinates=='e4':

return True

elif coordinates=='e5':

return False

elif coordinates=='e6':

return True

elif coordinates=='e7':

return False

elif coordinates=='e8':

return True

elif coordinates=='f1':

return True

elif coordinates=='f2':

return False

elif coordinates=='f3':

return True

elif coordinates=='f4':

return False

elif coordinates=='f5':

return True

elif coordinates=='f6':

return False

elif coordinates=='f7':

return True

elif coordinates=='f8':

return False

elif coordinates=='g1':

return False

elif coordinates=='g2':

return True

elif coordinates=='g3':

return False

elif coordinates=='g4':

return True

elif coordinates=='g5':

return False

elif coordinates=='g6':

return True

elif coordinates=='g7':

return False

elif coordinates=='g8':

return True

elif coordinates=='h1':

return True

elif coordinates=='h2':

return False

elif coordinates=='h3':

return True

elif coordinates=='h4':

return False

elif coordinates=='h5':

return True

elif coordinates=='h6':

return False

elif coordinates=='h7':

return True

elif coordinates=='h8':

return False

**1816. Truncate Sentence**

A **sentence** is a list of words that are separated by a single space with no leading or trailing spaces. Each of the words consists of **only** uppercase and lowercase English letters (no punctuation).

* For example, "Hello World", "HELLO", and "hello world hello world" are all sentences.

You are given a sentence s​​​​​​ and an integer k​​​​​​. You want to **truncate** s​​​​​​ such that it contains only the **first** k​​​​​​ words. Return s​​​​*​​ after****truncating****it.*

**Example 1:**

**Input:** s = "Hello how are you Contestant", k = 4

**Output:** "Hello how are you"

**Explanation:**

The words in s are ["Hello", "how" "are", "you", "Contestant"].

The first 4 words are ["Hello", "how", "are", "you"].

Hence, you should return "Hello how are you".

My sol:

class Solution:

def truncateSentence(self, s: str, k: int) -> str:

ans=""

a=[]

a=s.split()

b=a[:k]

ans = " ".join(b)

return ans

**1817. Finding the Users Active Minutes**

**( array as key in dictionary and id as value, reverse )**

You are given the logs for users' actions on LeetCode, and an integer k. The logs are represented by a 2D integer array logs where each logs[i] = [IDi, timei] indicates that the user with IDi performed an action at the minute timei.

**Multiple users** can perform actions simultaneously, and a single user can perform **multiple actions** in the same minute.

The **user active minutes (UAM)** for a given user is defined as the **number of unique minutes** in which the user performed an action on LeetCode. A minute can only be counted once, even if multiple actions occur during it.

You are to calculate a **1-indexed** array answer of size k such that, for each j (1 <= j <= k), answer[j] is the **number of users** whose **UAM** equals j.

Return *the array*answer*as described above*.

**Example 1:**

**Input:** logs = [[0,5],[1,2],[0,2],[0,5],[1,3]], k = 5

**Output:** [0,2,0,0,0]

**Explanation:**

The user with ID=0 performed actions at minutes 5, 2, and 5 again. Hence, they have a UAM of 2 (minute 5 is only counted once).

The user with ID=1 performed actions at minutes 2 and 3. Hence, they have a UAM of 2.

Since both users have a UAM of 2, answer[2] is 2, and the remaining answer[j] values are 0.

**My sol:**

class Solution:

def findingUsersActiveMinutes(self, logs: List[List[int]], k: int) -> List[int]:

d={}

for x in logs:

if x[0] in d:

d[x[0]].append(x[1])

else:

d[x[0]]=[x[1]]

for x,y in d.items():

d[x]=len(set(y))

p={}

for key,value in d.items():

if value in p:

p[value].append(key)

else:

p[value]=[key]

for keys,v in p.items():

p[keys]=len(v)

res=[0 for i in range(k)]

for keys,v in p.items():

res[keys-1]=v

return res

**1822. Sign of the Product of an Array**

There is a function signFunc(x) that returns:

* 1 if x is positive.
* -1 if x is negative.
* 0 if x is equal to 0.

You are given an integer array nums. Let product be the product of all values in the array nums.

Return signFunc(product).

**Example 1:**

**Input:** nums = [-1,-2,-3,-4,3,2,1]

**Output:** 1

**Explanation:** The product of all values in the array is 144, and signFunc(144) = 1

**My sol:**

class Solution:

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

prod=1

for x in nums:

prod=prod\*x

if prod>0:

return 1

elif prod<0:

return -1

else:

return 0

**1827. Minimum Operations to Make the Array Increasing**

You are given an integer array nums (**0-indexed**). In one operation, you can choose an element of the array and increment it by 1.

* For example, if nums = [1,2,3], you can choose to increment nums[1] to make nums = [1,**3**,3].

Return *the****minimum****number of operations needed to make* nums ***strictly******increasing****.*

An array nums is **strictly increasing** if nums[i] < nums[i+1] for all 0 <= i < nums.length - 1. An array of length 1 is trivially strictly increasing.

**Example 1:**

**Input:** nums = [1,1,1]

**Output:** 3

**Explanation:** You can do the following operations:

1) Increment nums[2], so nums becomes [1,1,**2**].

2) Increment nums[1], so nums becomes [1,**2**,2].

3) Increment nums[2], so nums becomes [1,2,**3**].

My sol:

class Solution:

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

count=0

first=sum(nums)

for x in range(0,len(nums)-1):

if nums[x+1]>nums[x]:

pass

else:

nums[x+1]=nums[x]+1

return (sum(nums)-first)

**1828. Queries on Number of Points Inside a Circle**

You are given an array points where points[i] = [xi, yi] is the coordinates of the ith point on a 2D plane. Multiple points can have the **same** coordinates.

You are also given an array queries where queries[j] = [xj, yj, rj] describes a circle centered at (xj, yj) with a radius of rj.

For each query queries[j], compute the number of points **inside** the jth circle. Points **on the border** of the circle are considered **inside**.

Return *an array*answer*, where*answer[j]*is the answer to the*jth*query*.

**Example 1:**

Diagram

Description automatically generated

**Input:** points = [[1,3],[3,3],[5,3],[2,2]], queries = [[2,3,1],[4,3,1],[1,1,2]]

**Output:** [3,2,2]

**Explanation:** The points and circles are shown above.

queries[0] is the green circle, queries[1] is the red circle, and queries[2] is the blue circle.

My sol:

class Solution:

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

count=0

final=[]

for q in queries:

for p in points:

if ((p[0]-q[0])\*\*2 + (p[1]-q[1])\*\*2) <= (q[2])\*\*2:

#print(p)

count=count+1

else:

pass

final.append(count)

count=0

return final

**1832. Check if the Sentence Is Pangram**

A **pangram** is a sentence where every letter of the English alphabet appears at least once.

Given a string sentence containing only lowercase English letters, returntrue*if*sentence*is a****pangram****, or*false*otherwise.*

**Example 1:**

**Input:** sentence = "thequickbrownfoxjumpsoverthelazydog"

**Output:** true

**Explanation:** sentence contains at least one of every letter of the English alphabet.

My sol:

class Solution:

def checkIfPangram(self, sentence: str) -> bool:

alp=["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"]

temp=1

for x in alp:

if x in sentence:

temp=1

else:

temp=2

break

print(temp)

if temp==1:

return 1==1

else:

return 1==2

**1833. Maximum Ice Cream Bars**It is a sweltering summer day, and a boy wants to buy some ice cream bars.

At the store, there are n ice cream bars. You are given an array costs of length n, where costs[i] is the price of the ith ice cream bar in coins. The boy initially has coins coins to spend, and he wants to buy as many ice cream bars as possible.

Return *the****maximum****number of ice cream bars the boy can buy with*coins*coins.*

**Note:** The boy can buy the ice cream bars in any order.

**Example 1:**

**Input:** costs = [1,3,2,4,1], coins = 7

**Output:** 4

**Explanation:** The boy can buy ice cream bars at indices 0,1,2,4 for a total price of 1 + 3 + 2 + 1 = 7.

My sol:

class Solution:

def maxIceCream(self, costs: List[int], coins: int) -> int:

count=0

f=sorted(costs)

for x in f:

if coins>=x:

count=count+1

coins=coins-x

else:

break

return count

**1836. Remove Duplicates From an Unsorted Linked List**

**(linked list as array and reconvert logic)**

Given the head of a linked list, find all the values that appear **more than once** in the list and delete the nodes that have any of those values.

Return *the linked list after the deletions.*

**Example 1:**

Diagram

Description automatically generated

**Input:** head = [1,2,3,2]

**Output:** [1,3]

**Explanation:** 2 appears twice in the linked list, so all 2's should be deleted. After deleting all 2's, we are left with [1,3].

**My sol:**

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

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

dummy=ListNode()

p=dummy

cur=head

temp=[]

while cur:

temp.append(cur.val)

cur=cur.next

d=collections.Counter(temp)

for x in temp:

if d[x]==1:

p.next=ListNode(x)

p=p.next

return dummy.next

**1837. Sum of Digits in Base K**

Given an integer n (in base 10) and a base k, return *the****sum****of the digits of*n***after****converting*n*from base*10*to base*k.

After converting, each digit should be interpreted as a base 10 number, and the sum should be returned in base 10.

**Example 1:**

**Input:** n = 34, k = 6

**Output:** 9

**Explanation:** 34 (base 10) expressed in base 6 is 54. 5 + 4 = 9.

My sol:

import numpy as np

class Solution:

def sumBase(self, n: int, k: int) -> int:

a=np.base\_repr(n, base=k)

print(a)

count=0

for x in a:

count+=int(x)

return count

**1844. Replace All Digits with Characters**

You are given a **0-indexed** string s that has lowercase English letters in its **even** indices and digits in its **odd** indices.

There is a function shift(c, x), where c is a character and x is a digit, that returns the xth character after c.

* For example, shift('a', 5) = 'f' and shift('x', 0) = 'x'.

For every **odd** index i, you want to replace the digit s[i] with shift(s[i-1], s[i]).

Return s*after replacing all digits. It is****guaranteed****that*shift(s[i-1], s[i])*will never exceed*'z'.

**Example 1:**

**Input:** s = "a1c1e1"

**Output:** "abcdef"

**Explanation:** The digits are replaced as follows:

- s[1] -> shift('a',1) = 'b'

- s[3] -> shift('c',1) = 'd'

- s[5] -> shift('e',1) = 'f'

My sol:

class Solution:

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

temp=['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']

for x in range(0,len(s)):

if s[x] in ['0','1','2','3','4','5','6','7','8','9']:

s=s.replace(s[x],temp[temp.index(s[x-1])+int(s[x])],1)

return s

**1848. Minimum Distance to the Target Element**

Given an integer array nums **(0-indexed)** and two integers target and start, find an index i such that nums[i] == target and abs(i - start) is **minimized**. Note that abs(x) is the absolute value of x.

Return abs(i - start).

It is **guaranteed** that target exists in nums.

**Example 1:**

**Input:** nums = [1,2,3,4,5], target = 5, start = 3

**Output:** 1

**Explanation:** nums[4] = 5 is the only value equal to target, so the answer is abs(4 - 3) = 1.

**My sol:**

class Solution:

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

minval =10000

for x in range(0,len(nums)):

if nums[x]==target and abs(x-start)<minval:

minval=abs(x-start)

return minval

1851-1900 ( $$$$$ 12)

**1852. Distinct Numbers in Each Subarray ( subarray main logic sliding window o(N))**

Given an integer array nums and an integer k, you are asked to construct the array ans of size n-k+1 where ans[i] is the number of **distinct** numbers in the subarray nums[i:i+k-1] = [nums[i], nums[i+1], ..., nums[i+k-1]].

Return *the array*ans.

**Example 1:**

**Input:** nums = [1,2,3,2,2,1,3], k = 3

**Output:** [3,2,2,2,3]

**Explanation:** The number of distinct elements in each subarray goes as follows:

- nums[0:2] = [1,2,3] so ans[0] = 3

- nums[1:3] = [2,3,2] so ans[1] = 2

- nums[2:4] = [3,2,2] so ans[2] = 2

- nums[3:5] = [2,2,1] so ans[3] = 2

- nums[4:6] = [2,1,3] so ans[4] = 3

My sol:

class Solution:

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

i=0

final=[]

d=collections.Counter(nums[i:i+k])

final.append(len(d))

i+=1

while i<=len(nums)-k:

# print(nums[i:i+k])

# print(nums[i])

# print(nums[i+k-1])

if nums[i+k-1] in d:

d[nums[i+k-1]]+=1

else:

d[nums[i+k-1]]=1

d[nums[i-1]]-=1

if d[nums[i-1]]==0:

del d[nums[i-1]]

i+=1

final.append(len(d))

return final

**1854. Maximum Population Year**

You are given a 2D integer array logs where each logs[i] = [birthi, deathi] indicates the birth and death years of the ith person.

The **population** of some year x is the number of people alive during that year. The ith person is counted in year x's population if x is in the **inclusive** range [birthi, deathi - 1]. Note that the person is **not** counted in the year that they die.

Return *the****earliest****year with the****maximum population***.

**Example 1:**

**Input:** logs = [[1993,1999],[2000,2010]]

**Output:** 1993

**Explanation:** The maximum population is 1, and 1993 is the earliest year with this population.

My sol:

class Solution:

def maximumPopulation(self, logs: List[List[int]]) -> int:

dict={}

a=[]

b=[]

for x in logs:

for y in range(x[0],x[1]):

if y in dict:

dict[y]+=1

else:

dict[y]=1

maxval=0

for key,values in dict.items():

if values>maxval:

maxval=values

for key,values in dict.items():

if values==maxval:

a.append(key)

# if values==1:

# b.append(key)

# if a==[]:

# return sorted(b)[0]

return sorted(a)[0]

**1859. Sorting the Sentence**

A **sentence** is a list of words that are separated by a single space with no leading or trailing spaces. Each word consists of lowercase and uppercase English letters.

A sentence can be **shuffled** by appending the **1-indexed word position** to each word then rearranging the words in the sentence.

* For example, the sentence "This is a sentence" can be shuffled as "sentence4 a3 is2 This1" or "is2 sentence4 This1 a3".

Given a **shuffled sentence** s containing no more than 9 words, reconstruct and return *the original sentence*.

**Example 1:**

**Input:** s = "is2 sentence4 This1 a3"

**Output:** "This is a sentence"

**Explanation:** Sort the words in s to their original positions "This1 is2 a3 sentence4", then remove the numbers.

My sol:

class Solution:

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

f=""

a=s.split()

print(a)

ans=[0 for x in range(0,len(a))]

for x in a:

ans[int(x[-1])-1]=x[:-1]

f=" ".join(ans)

return f

**1869. Longer Contiguous Segments of Ones than Zeros**

Given a binary string s, return true*if the****longest****contiguous segment of*1'*s is****strictly longer****than the****longest****contiguous segment of*0'*s in*s, or return false*otherwise*.

* For example, in s = "110100010" the longest continuous segment of 1s has length 2, and the longest continuous segment of 0s has length 3.

Note that if there are no 0's, then the longest continuous segment of 0's is considered to have a length 0. The same applies if there is no 1's.

**Example 1:**

**Input:** s = "1101"

**Output:** true

**Explanation:**

The longest contiguous segment of 1s has length 2: "1101"

The longest contiguous segment of 0s has length 1: "1101"

The segment of 1s is longer, so return true.

My sol:

class Solution:

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

maximum1=0

count1=0;

for x in range(0,len(s)):

if s[x] == '1':

count1=count1+1

if s[x]=='0' :

maximum1=max(maximum1,count1)

count1=0

maximum1=max(maximum1,count1)

maximum2=0

count2=0;

for x in range(0,len(s)):

if s[x] == '0':

count2=count2+1

if s[x]=='1':

maximum2=max(maximum2,count2)

count2=0

maximum2=max(maximum2,count2)

return bool(maximum1>maximum2)

**1874. Minimize Product Sum of Two Arrays**

The **product sum**of two equal-length arrays a and b is equal to the sum of a[i] \* b[i] for all 0 <= i < a.length (**0-indexed**).

* For example, if a = [1,2,3,4] and b = [5,2,3,1], the **product sum** would be 1\*5 + 2\*2 + 3\*3 + 4\*1 = 22.

Given two arrays nums1 and nums2 of length n, return *the****minimum product sum****if you are allowed to****rearrange****the****order****of the elements in*nums1.

**Example 1:**

**Input:** nums1 = [5,3,4,2], nums2 = [4,2,2,5]

**Output:** 40

**Explanation:** We can rearrange nums1 to become [3,5,4,2]. The product sum of [3,5,4,2] and [4,2,2,5] is 3\*4 + 5\*2 + 4\*2 + 2\*5 = 40.

My sol:

class Solution:

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

a=sorted(nums1)

b=sorted(nums2)[::-1]

count=0

for i in range(0,len(nums1)):

count+=a[i]\*b[i]

return count

**1876. Substrings of Size Three with Distinct Characters**

**( Subarray sliding window logic so powerful compared to bruteforce)**

A string is **good** if there are no repeated characters.

Given a string s​​​​​, return *the number of****good substrings****of length****three****in*s​​​​​​.

Note that if there are multiple occurrences of the same substring, every occurrence should be counted.

A **substring** is a contiguous sequence of characters in a string.

**Example 1:**

**Input:** s = "xyzzaz"

**Output:** 1

**Explanation:** There are 4 substrings of size 3: "xyz", "yzz", "zza", and "zaz".

The only good substring of length 3 is "xyz".

My sol:

Optimized sliding window subarray

class Solution:

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

count=0

i=0

d=collections.Counter(s[i:i+3])

if len(d)==3:

count+=1

for i in range(1,len(s)-3+1):

if s[i+3-1] in d:

d[s[i+3-1]]+=1

else:

d[s[i+3-1]]=1

d[s[i-1]]-=1

if d[s[i-1]]==0:

del d[s[i-1]]

if len(d)==3:

count+=1

return count

**bruteforce:**

class Solution:

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

# count=0

# a=[s[i:j+1] for i in range(len(s)) for j in range(i,len(s))]

# for x in a:

# if len(x)==3 and len(Counter(x))==len(x):

# count+=1

# return count

**1877. Minimize Maximum Pair Sum in Array**

The **pair sum** of a pair (a,b) is equal to a + b. The **maximum pair sum** is the largest **pair sum** in a list of pairs.

* For example, if we have pairs (1,5), (2,3), and (4,4), the **maximum pair sum** would be max(1+5, 2+3, 4+4) = max(6, 5, 8) = 8.

Given an array nums of **even** length n, pair up the elements of nums into n / 2 pairs such that:

* Each element of nums is in **exactly one** pair, and
* The **maximum pair sum**is **minimized**.

Return *the minimized****maximum pair sum****after optimally pairing up the elements*.

**Example 1:**

**Input:** nums = [3,5,2,3]

**Output:** 7

**Explanation:** The elements can be paired up into pairs (3,3) and (5,2).

The maximum pair sum is max(3+3, 5+2) = max(6, 7) = 7.

My sol:

class Solution:

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

a=sorted(nums)

i=0

j=len(a)-1

temp=0

res=0

if len(a)%2!=0:

return -1

while i<j:

temp=a[i]+a[j]

res=max(temp,res)

i+=1

j-=1

return res

**1880. Check if Word Equals Summation of Two Words**

The **letter value** of a letter is its position in the alphabet **starting from 0** (i.e. 'a' -> 0, 'b' -> 1, 'c' -> 2, etc.).

The **numerical value** of some string of lowercase English letters s is the **concatenation** of the **letter values** of each letter in s, which is then **converted** into an integer.

* For example, if s = "acb", we concatenate each letter's letter value, resulting in "021". After converting it, we get 21.

You are given three strings firstWord, secondWord, and targetWord, each consisting of lowercase English letters 'a' through 'j' **inclusive**.

Return true *if the****summation****of the****numerical values****of*firstWord*and*secondWord*equals the****numerical value****of*targetWord*, or*false*otherwise.*

**Example 1:**

**Input:** firstWord = "acb", secondWord = "cba", targetWord = "cdb"

**Output:** true

**Explanation:**

The numerical value of firstWord is "acb" -> "021" -> 21.

The numerical value of secondWord is "cba" -> "210" -> 210.

The numerical value of targetWord is "cdb" -> "231" -> 231.

We return true because 21 + 210 == 231.

My sol:

class Solution:

def isSumEqual(self, firstWord: str, secondWord: str, targetWord: str) -> bool:

for x in firstWord:

if x =='a':

firstWord=firstWord.replace(x,'0')

if x =='b':

firstWord=firstWord.replace(x,'1')

if x =='c':

firstWord=firstWord.replace(x,'2')

if x =='d':

firstWord=firstWord.replace(x,'3')

if x =='e':

firstWord=firstWord.replace(x,'4')

if x =='f':

firstWord=firstWord.replace(x,'5')

if x =='g':

firstWord=firstWord.replace(x,'6')

if x =='h':

firstWord=firstWord.replace(x,'7')

if x =='i':

firstWord=firstWord.replace(x,'8')

if x =='j':

firstWord=firstWord.replace(x,'9')

for x in secondWord:

if x =='a':

secondWord=secondWord.replace(x,'0')

if x =='b':

secondWord=secondWord.replace(x,'1')

if x =='c':

secondWord=secondWord.replace(x,'2')

if x =='d':

secondWord=secondWord.replace(x,'3')

if x =='e':

secondWord=secondWord.replace(x,'4')

if x =='f':

secondWord=secondWord.replace(x,'5')

if x =='g':

secondWord=secondWord.replace(x,'6')

if x =='h':

secondWord=secondWord.replace(x,'7')

if x =='i':

secondWord=secondWord.replace(x,'8')

if x =='j':

secondWord=secondWord.replace(x,'9')

for x in targetWord:

if x =='a':

targetWord=targetWord.replace(x,'0')

if x =='b':

targetWord=targetWord.replace(x,'1')

if x =='c':

targetWord=targetWord.replace(x,'2')

if x =='d':

targetWord=targetWord.replace(x,'3')

if x =='e':

targetWord=targetWord.replace(x,'4')

if x =='f':

targetWord=targetWord.replace(x,'5')

if x =='g':

targetWord=targetWord.replace(x,'6')

if x =='h':

targetWord=targetWord.replace(x,'7')

if x =='i':

targetWord=targetWord.replace(x,'8')

if x =='j':

targetWord=targetWord.replace(x,'9')

print(int(firstWord))

print(int(secondWord))

print(int(targetWord))

return int(targetWord)== (int(firstWord) + int(secondWord))

**1886. Determine Whether Matrix Can Be Obtained By Rotation**

Given two n x n binary matrices mat and target, return true*if it is possible to make*mat*equal to*target*by****rotating***mat*in****90-degree increments****, or*false*otherwise.*

**Example 1:**

A picture containing text, clock

Description automatically generated

**Input:** mat = [[0,1],[1,0]], target = [[1,0],[0,1]]

**Output:** true

**Explanation:** We can rotate mat 90 degrees clockwise to make mat equal target.

**My sol:**

class Solution:

def findRotation(self, mat: List[List[int]], target: List[List[int]]) -> bool:

count=0

if mat==target:

return True

while count<3:

i=0

b=[]

a=[]

while i<len(mat[0]):

for x in mat:

a.append(x[i])

b.append(a[::-1])

a=[]

i+=1

mat[:]=b

print(mat)

count+=1

if mat==target:

return True

return False

**1893. Check if All the Integers in a Range Are Covered**

You are given a 2D integer array ranges and two integers left and right. Each ranges[i] = [starti, endi] represents an **inclusive** interval between starti and endi.

Return true *if each integer in the inclusive range* [left, right] *is covered by****at least one****interval in* ranges. Return false *otherwise*.

An integer x is covered by an interval ranges[i] = [starti, endi] if starti <= x <= endi.

**Example 1:**

**Input:** ranges = [[1,2],[3,4],[5,6]], left = 2, right = 5

**Output:** true

**Explanation:** Every integer between 2 and 5 is covered:

- 2 is covered by the first range.

- 3 and 4 are covered by the second range.

- 5 is covered by the third range.

**My sol:**

class Solution:

def isCovered(self, ranges: List[List[int]], left: int, right: int) -> bool:

flag=0

temp=[]

for i in range(0,len(ranges)):

for y in range(ranges[i][0],ranges[i][1]+1):

temp.append(y)

for i in range(left,right+1):

if i in temp:

flag=0

else:

flag=1

break

if flag==0:

return 1==1

else:

return 1==2

**1894. Find the Student that Will Replace the Chalk**

There are n students in a class numbered from 0 to n - 1. The teacher will give each student a problem starting with the student number 0, then the student number 1, and so on until the teacher reaches the student number n - 1. After that, the teacher will restart the process, starting with the student number 0 again.

You are given a **0-indexed** integer array chalk and an integer k. There are initially k pieces of chalk. When the student number i is given a problem to solve, they will use chalk[i] pieces of chalk to solve that problem. However, if the current number of chalk pieces is **strictly less** than chalk[i], then the student number i will be asked to **replace** the chalk.

Return *the****index****of the student that will****replace****the chalk*.

**Example 1:**

**Input:** chalk = [5,1,5], k = 22

**Output:** 0

**Explanation:** The students go in turns as follows:

- Student number 0 uses 5 chalk, so k = 17.

- Student number 1 uses 1 chalk, so k = 16.

- Student number 2 uses 5 chalk, so k = 11.

- Student number 0 uses 5 chalk, so k = 6.

- Student number 1 uses 1 chalk, so k = 5.

- Student number 2 uses 5 chalk, so k = 0.

Student number 0 does not have enough chalk, so they will have to replace it.

**My sol:**

class Solution:

def chalkReplacer(self, chalk: List[int], k: int) -> int:

s=sum(chalk);

md = float(k % s)

print(s)

print(md)

if md == 0:

return 0

else:

for i in range(0,len(chalk)):

if md-chalk[i]< 0:

return i;

else:

md=md-chalk[i];

**1897. Redistribute Characters to Make All Strings Equal**

You are given an array of strings words (**0-indexed**).

In one operation, pick two **distinct** indices i and j, where words[i] is a non-empty string, and move **any** character from words[i] to **any** position in words[j].

Return true *if you can make****every****string in*words***equal****using****any****number of operations*,*and*false *otherwise*.

**Example 1:**

**Input:** words = ["abc","aabc","bc"]

**Output:** true

**Explanation:** Move the first 'a' in words[1] to the front of words[2],

to make words[1] = "abc" and words[2] = "abc".

All the strings are now equal to "abc", so return true.

**My sol:**

class Solution:

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

dict={}

for x in words:

for y in x:

if y in dict:

dict[y]+=1

else:

dict[y]=1

for k,v in dict.items():

if v%len(words)!=0:

return False

return True

1901-1950 ($$$$$ 12)

**1901. Find a Peak Element II**

**peak** element in a 2D grid is an element that is **strictly greater** than all of its **adjacent**neighbors to the left, right, top, and bottom.

Given a **0-indexed** m x n matrix mat where **no two adjacent cells are equal**, find **any** peak element mat[i][j] and return *the length 2 array*[i,j].

You may assume that the entire matrix is surrounded by an **outer perimeter** with the value -1 in each cell.

You must write an algorithm that runs in O(m log(n)) or O(n log(m)) time.

**Example 1:**

A picture containing furniture, file

Description automatically generated

**Input:** mat = [[1,4],[3,2]]

**Output:** [0,1]

**Explanation:** Both 3 and 4 are peak elements so [1,0] and [0,1] are both acceptable answers.

My sol:

class Solution:

def findPeakGrid(self, mat: List[List[int]]) -> List[int]:

a=[-1] \* (len(mat[0])+2)

# print(a)

temp=[a]

r=[-1]

for x in mat:

r=[-1]

for y in x:

r.append(y)

r.append(-1)

temp.append(r)

temp.append(a)

# print(temp)

res=[]

for x in range(1,len(temp)-1):

for y in range(1,len(temp[x])-1):

if temp[x][y]>temp[x][y-1] and temp[x][y]>temp[x-1][y] and temp[x][y]>temp[x+1][y] and temp[x][y]>temp[x][y+1]:

return [x-1,y-1]

return res

**1903. Largest Odd Number in String**

You are given a string num, representing a large integer. Return *the****largest-valued odd****integer (as a string) that is a****non-empty substring****of*num*, or an empty string*""*if no odd integer exists*.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** num = "52"

**Output:** "5"

**Explanation:** The only non-empty substrings are "5", "2", and "52". "5" is the only odd number.

My sol:

class Solution:

def largestOddNumber(self, num: str) -> str:

n=str(num)

for x in range(len(n)-1,-1,-1):

if int(n[x])%2 == 1:

return n[:x+1]

return ""

**1909. Remove One Element to Make the Array Strictly Increasing**

Given a **0-indexed** integer array nums, return true *if it can be made****strictly increasing****after removing****exactly one****element, or*false*otherwise. If the array is already strictly increasing, return*true.

The array nums is **strictly increasing** if nums[i - 1] < nums[i] for each index (1 <= i < nums.length).

**Example 1:**

**Input:** nums = [1,2,10,5,7]

**Output:** true

**Explanation:** By removing 10 at index 2 from nums, it becomes [1,2,5,7].

[1,2,5,7] is strictly increasing, so return true.

My sol:

class Solution:

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

n=nums

for x in range(0,len(nums)):

d=n[x]

nums.pop(x)

# print(nums)

if sorted(nums)==nums and len(list(set(nums)))==len(nums):

return True

nums.insert(x,d)

# print(nums)

return False

**1910. Remove All Occurrences of a Substring**

Given two strings s and part, perform the following operation on s until **all** occurrences of the substring part are removed:

* Find the **leftmost** occurrence of the substring part and **remove** it from s.

Return s*after removing all occurrences of*part.

A **substring** is a contiguous sequence of characters in a string.

**Example 1:**

**Input:** s = "daabcbaabcbc", part = "abc"

**Output:** "dab"

**Explanation**: The following operations are done:

- s = "da**abc**baabcbc", remove "abc" starting at index 2, so s = "dabaabcbc".

- s = "daba**abc**bc", remove "abc" starting at index 4, so s = "dababc".

- s = "dab**abc**", remove "abc" starting at index 3, so s = "dab".

Now s has no occurrences of "abc".

My sol:

class Solution:

def removeOccurrences(self, s: str, part: str) -> str:

while part in s:

s=s.replace(part,"",1)

return s

**1913. Maximum Product Difference Between Two Pairs**

The **product difference** between two pairs (a, b) and (c, d) is defined as (a \* b) - (c \* d).

* For example, the product difference between (5, 6) and (2, 7) is (5 \* 6) - (2 \* 7) = 16.

Given an integer array nums, choose four **distinct** indices w, x, y, and z such that the **product difference** between pairs (nums[w], nums[x]) and (nums[y], nums[z]) is **maximized**.

Return *the****maximum****such product difference*.

**Example 1:**

**Input:** nums = [5,6,2,7,4]

**Output:** 34

**Explanation:** We can choose indices 1 and 3 for the first pair (6, 7) and indices 2 and 4 for the second pair (2, 4).

The product difference is (6 \* 7) - (2 \* 4) = 34.

My sol:

class Solution:

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

n=sorted(nums)

print(n)

sum1=0

sum2=0

sum1=n[0]\*n[1]

sum2=n[len(n)-1]\*n[len(n)-2]

return sum2-sum1

**1920. Build Array from Permutation**

Given a **zero-based permutation** nums (**0-indexed**), build an array ans of the **same length** where ans[i] = nums[nums[i]] for each 0 <= i < nums.length and return it.

A **zero-based permutation** nums is an array of **distinct** integers from 0 to nums.length - 1 (**inclusive**).

**Example 1:**

**Input:** nums = [0,2,1,5,3,4]

**Output:** [0,1,2,4,5,3]

**Explanation:** The array ans is built as follows:

ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]], nums[nums[4]], nums[nums[5]]]

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

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

My sol:

class Solution:

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

output=[]

for x in nums:

output.append(nums[x])

return output

**1925. Count Square Sum Triples**

A **square triple** (a,b,c) is a triple where a, b, and c are **integers** and a2 + b2 = c2.

Given an integer n, return *the number of****square triples****such that*1 <= a, b, c <= n.

**Example 1:**

**Input:** n = 5

**Output:** 2

**Explanation**: The square triples are (3,4,5) and (4,3,5).

My sol:

import math

class Solution:

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

temp=[i for i in range(1,n+1)]

t=collections.Counter(temp)

count=0

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

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

if math.sqrt(abs((x+y)\*(x-y))) != x and math.sqrt(abs((x+y)\*(x-y))) != y and math.sqrt(abs((x+y)\*(x-y))) in t:

count+=1

return count//2

**1929. Concatenation of Array**

Given an integer array nums of length n, you want to create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (**0-indexed**).

Specifically, ans is the **concatenation** of two nums arrays.

Return *the array*ans.

**Example 1:**

**Input:** nums = [1,2,1]

**Output:** [1,2,1,1,2,1]

**Explanation:** The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]]

- ans = [1,2,1,1,2,1]

My sol:

class Solution:

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

ans=[]

ans.extend(nums)

ans.extend(nums)

return ans

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

There is a malfunctioning keyboard where some letter keys do not work. All other keys on the keyboard work properly.

Given a string text of words separated by a single space (no leading or trailing spaces) and a string brokenLetters of all **distinct** letter keys that are broken, return *the****number of words****in* text *you can fully type using this keyboard*.

**Example 1:**

**Input:** text = "hello world", brokenLetters = "ad"

**Output:** 1

**Explanation:** We cannot type "world" because the 'd' key is broken.

My sol:

class Solution:

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

textlist=text.split();

count=0

flag=0

print(textlist)

if len(brokenLetters)==0:

return len(textlist)

for x in textlist:

for y in brokenLetters:

if y in x:

flag=0

break

else:

flag=1

if flag==0:

count+=1

flag=0

return len(textlist)-count

**1940. Longest Common Subsequence Between Sorted Arrays**

Given an array of integer arrays arrays where each arrays[i] is sorted in **strictly increasing** order, return *an integer array representing the****longest common subsequence****between****all****the arrays*.

A **subsequence** is a sequence that can be derived from another sequence by deleting some elements (possibly none) without changing the order of the remaining elements.

**Example 1:**

**Input:** arrays = [[1,3,4],

[1,4,7,9]]

**Output:** [1,4]

**Explanation:** The longest common subsequence in the two arrays is [1,4].

My sol:

class Solution:

def longestCommonSubsequence(self, arrays: List[List[int]]) -> List[int]:

flag=0

a=[]

for y in arrays[0]:

for x in range(1,len(arrays)):

if y in arrays[x]:

flag=0

else:

flag=1

break

if flag==0 and y not in a:

a.append(y)

return a

**1941. Check if All Characters Have Equal Number of Occurrences**

Given a string s, return true*if*s*is a****good****string, or*false*otherwise*.

A string s is **good** if **all** the characters that appear in s have the **same** number of occurrences (i.e., the same frequency).

**Example 1:**

**Input:** s = "abacbc"

**Output:** true

**Explanation:** The characters that appear in s are 'a', 'b', and 'c'. All characters occur 2 times in s.

My sol:

class Solution:

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

flag=0

a=[]

dict={}

for x in s:

if x in dict:

dict[x]+=1

else:

dict[x]=1

for key,value in dict.items():

a.append(value)

for x in a:

if a[0]==x:

flag=0

else:

flag=1

break

return bool(flag==0)

**1945. Sum of Digits of String After Convert**

You are given a string s consisting of lowercase English letters, and an integer k.

First, **convert** s into an integer by replacing each letter with its position in the alphabet (i.e., replace 'a' with 1, 'b' with 2, ..., 'z' with 26). Then, **transform** the integer by replacing it with the **sum of its digits**. Repeat the **transform** operation k**times** in total.

For example, if s = "zbax" and k = 2, then the resulting integer would be 8 by the following operations:

* **Convert**: "zbax" ➝ "(26)(2)(1)(24)" ➝ "262124" ➝ 262124
* **Transform #1**: 262124 ➝ 2 + 6 + 2 + 1 + 2 + 4 ➝ 17
* **Transform #2**: 17 ➝ 1 + 7 ➝ 8

Return *the resulting integer after performing the operations described above*.

**Example 1:**

**Input:** s = "iiii", k = 1

**Output:** 36

**Explanation:** The operations are as follows:

- Convert: "iiii" ➝ "(9)(9)(9)(9)" ➝ "9999" ➝ 9999

- Transform #1: 9999 ➝ 9 + 9 + 9 + 9 ➝ 36

Thus the resulting integer is 36.

My sol:

class Solution:

def getLucky(self, s: str, k: int) -> int:

temp=""

count=0

for x in s:

if x=='a':

temp=temp+'1'

elif x=='b':

temp=temp+'2'

elif x=='c':

temp=temp+'3'

elif x=='d':

temp=temp+'4'

elif x=='e':

temp=temp+'5'

elif x=='f':

temp=temp+'6'

elif x=='g':

temp=temp+'7'

elif x=='h':

temp=temp+'8'

elif x=='i':

temp=temp+'9'

elif x=='j':

temp=temp+'10'

elif x=='k':

temp=temp+'11'

elif x=='l':

temp=temp+'12'

elif x=='m':

temp=temp+'13'

elif x=='n':

temp=temp+'14'

elif x=='o':

temp=temp+'15'

elif x=='p':

temp=temp+'16'

elif x=='q':

temp=temp+'17'

elif x=='r':

temp=temp+'18'

elif x=='s':

temp=temp+'19'

elif x=='t':

temp=temp+'20'

elif x=='u':

temp=temp+'21'

elif x=='v':

temp=temp+'22'

elif x=='w':

temp=temp+'23'

elif x=='x':

temp=temp+'24'

elif x=='y':

temp=temp+'25'

elif x=='z':

temp=temp+'26'

while k>=1:

count=0

for x in temp:

count+=int(x)

temp=str(count)

k-=1

return count

1951-2000 ($$$$$ 12)

**1952. Three Divisors**

Given an integer n, return true*if*n*has****exactly three positive divisors****. Otherwise, return*false.

An integer m is a **divisor** of n if there exists an integer k such that n = k \* m.

**Example 1:**

**Input:** n = 2

**Output:** false

**Explantion:** 2 has only two divisors: 1 and 2.

My sol:

class Solution:

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

count =2;

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

if n % i == 0 and count <=3:

count+=1;

return count ==3

**1957. Delete Characters to Make Fancy String**

A **fancy string** is a string where no **three** **consecutive** characters are equal.

Given a string s, delete the **minimum** possible number of characters from s to make it **fancy**.

Return *the final string after the deletion*. It can be shown that the answer will always be **unique**.

**Example 1:**

**Input:** s = "leeetcode"

**Output:** "leetcode"

**Explanation:**

Remove an 'e' from the first group of 'e's to create "leetcode".

No three consecutive characters are equal, so return "leetcode".

My sol:

class Solution:

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

a=list(set(s))

for x in a:

while x\*3 in s:

s=s.replace(x\*3,x\*2)

return s

**1961. Check If String Is a Prefix of Array**

Given a string s and an array of strings words, determine whether s is a **prefix string** of words.

A string s is a **prefix string** of words if s can be made by concatenating the first k strings in words for some **positive** k no larger than words.length.

Return true*if*s*is a****prefix string****of*words*, or*false*otherwise*.

**Example 1:**

**Input:** s = "iloveleetcode", words = ["i","love","leetcode","apples"]

**Output:** true

**Explanation:**

s can be made by concatenating "i", "love", and "leetcode" together.

My sol:

class Solution:

def isPrefixString(self, s: str, words: List[str]) -> bool:

a=[]

count=0

flag=0

for x in words:

if x in s:

print(s.find(x))

a.append(s.find(x))

counwt+=len(x)

else:

break

if sorted(a)==a and count==len(s):

return True

else:

return False

**1967. Number of Strings That Appear as Substrings in Word**

Given an array of strings patterns and a string word, return *the****number****of strings in*patterns*that exist as a****substring****in*word.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** patterns = ["a","abc","bc","d"], word = "abc"

**Output:** 3

**Explanation:**

- "a" appears as a substring in "abc".

- "abc" appears as a substring in "abc".

- "bc" appears as a substring in "abc".

- "d" does not appear as a substring in "abc".

3 of the strings in patterns appear as a substring in word.

My sol:

class Solution:

def numOfStrings(self, patterns: List[str], word: str) -> int:

count=0

for x in patterns:

if x in word:

count+=1

return count

**1971. Find if Path Exists in Graph (bfs)**

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to n - 1 (**inclusive**). The edges in the graph are represented as a 2D integer array edges, where each edges[i] = [ui, vi] denotes a bi-directional edge between vertex ui and vertex vi. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex source to vertex destination.

Given edges and the integers n, source, and destination, return true*if there is a****valid path****from*source*to*destination*, or*false*otherwise.*

**Example 1:**

A picture containing text, clipart

Description automatically generated

**Input:** n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2

**Output:** true

**Explanation:** There are two paths from vertex 0 to vertex 2:

- 0 → 1 → 2

- 0 → 2

My sol:

class Solution:

def validPath(self, n: int, edges: List[List[int]], start: int, end: int) -> bool:

if start == end:

return True

g = defaultdict(list)

def createGraph(edges):

for edge in edges:

g[edge[0]].append(edge[1])

g[edge[1]].append(edge[0])

createGraph(edges)

print(g)

visited = []

deq = collections.deque([start])

while deq:

child = deq.popleft()

if end in g[child]:

return True

if child not in visited:

visited.append(child)

for children in g[child]:

deq.appendleft(children)

return False

**1973. Count Nodes Equal to Sum of Descendants**

Given the root of a binary tree, return *the number of nodes where the value of the node is equal to the****sum****of the values of its descendants*.

A **descendant** of a node x is any node that is on the path from node x to some leaf node. The sum is considered to be 0 if the node has no descendants.

**Example 1:**

A picture containing text, clock

Description automatically generated

**Input:** root = [10,3,4,2,1]

**Output:** 2

**Explanation:**

For the node with value 10: The sum of its descendants is 3+4+2+1 = 10.

For the node with value 3: The sum of its descendants is 2+1 = 3.

My sol:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

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

def fn(node):

nonlocal ans

if not node:

return 0

sm = fn(node.left) + fn(node.right)

if sm == node.val:

ans += 1

return sm + node.val

ans = 0

fn(root)

return ans

**1974. Minimum Time to Type Word Using Special Typewriter**

There is a special typewriter with lowercase English letters 'a' to 'z' arranged in a **circle** with a **pointer**. A character can **only** be typed if the pointer is pointing to that character. The pointer is **initially** pointing to the character 'a'.

Chart, pie chart

Description automatically generated

Each second, you may perform one of the following operations:

* Move the pointer one character **counterclockwise** or **clockwise**.
* Type the character the pointer is **currently** on.

Given a string word, return the**minimum** number of seconds to type out the characters in word.

**Example 1:**

**Input:** word = "abc"

**Output:** 5

**Explanation:**

The characters are printed as follows:

- Type the character 'a' in 1 second since the pointer is initially on 'a'.

- Move the pointer clockwise to 'b' in 1 second.

- Type the character 'b' in 1 second.

- Move the pointer clockwise to 'c' in 1 second.

- Type the character 'c' in 1 second.

My sol:

class Solution:

def minTimeToType(self, word: str) -> int:

dict={}

dict['a']=1

dict['b']=2

dict['c']=3

dict['d']=4

dict['e']=5

dict['f']=6

dict['g']=7

dict['h']=8

dict['i']=9

dict['j']=10

dict['k']=11

dict['l']=12

dict['m']=13

dict['n']=14

dict['o']=15

dict['p']=16

dict['q']=17

dict['r']=18

dict['s']=19

dict['t']=20

dict['u']=21

dict['v']=22

dict['w']=23

dict['x']=24

dict['y']=25

dict['z']=26

count=0

po='a'

for x in word:

count+=min(abs(dict[po]-dict[x]),26-(abs(dict[x]-dict[po])))+1

po=x

return count

**1979. Find Greatest Common Divisor of Array**

Given an integer array nums, return*the****greatest common divisor****of the smallest number and largest number in*nums.

The **greatest common divisor** of two numbers is the largest positive integer that evenly divides both numbers.

**Example 1:**

**Input:** nums = [2,5,6,9,10]

**Output:** 2

**Explanation:**

The smallest number in nums is 2.

The largest number in nums is 10.

The greatest common divisor of 2 and 10 is 2.

My sol:

class Solution:

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

s= min(nums)

l= max(nums)

def hcfnaive(a,b):

if(b==0):

return a

else:

return hcfnaive(b,a%b)

return hcfnaive(l,s)

**1984. Minimum Difference Between Highest and Lowest of K Scores**

You are given a **0-indexed** integer array nums, where nums[i] represents the score of the ith student. You are also given an integer k.

Pick the scores of any k students from the array so that the **difference** between the **highest** and the **lowest** of the k scores is **minimized**.

Return *the****minimum****possible difference*.

**Example 1:**

**Input:** nums = [90], k = 1

**Output:** 0

**Explanation:** There is one way to pick score(s) of one student:

- [**90**]. The difference between the highest and lowest score is 90 - 90 = 0.

The minimum possible difference is 0.

My sol:

class Solution:

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

a=sorted(nums)

i=0

res=100000000000

if len(nums)==1:

return 0

while i<len(a)-1:

if len(a[i:i+k])==k:

temp=max(a[i:i+k])-min(a[i:i+k])

res=min(res,temp)

i+=1

return res

**1985. Find the Kth Largest Integer in the Array**

You are given an array of strings nums and an integer k. Each string in nums represents an integer without leading zeros.

Return *the string that represents the*kth***largest integer****in*nums.

**Note**: Duplicate numbers should be counted distinctly. For example, if nums is ["1","2","2"], "2" is the first largest integer, "2" is the second-largest integer, and "1" is the third-largest integer.

**Example 1:**

**Input:** nums = ["3","6","7","10"], k = 4

**Output:** "3"

**Explanation:**

The numbers in nums sorted in non-decreasing order are ["3","6","7","10"].

The 4th largest integer in nums is "3".

My sol:

class Solution:

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

a=[]

for x in nums:

a.append(int(x))

b=sorted(a)

c=b[::-1][k-1:k]

return str(c[0])

**1991. Find the Middle Index in Array ( kiran goldman question)**

Given a **0-indexed** integer array nums, find the **leftmost** middleIndex (i.e., the smallest amongst all the possible ones).

A middleIndex is an index where nums[0] + nums[1] + ... + nums[middleIndex-1] == nums[middleIndex+1] + nums[middleIndex+2] + ... + nums[nums.length-1].

If middleIndex == 0, the left side sum is considered to be 0. Similarly, if middleIndex == nums.length - 1, the right side sum is considered to be 0.

Return *the****leftmost***middleIndex*that satisfies the condition, or*-1*if there is no such index*.

**Example 1:**

**Input:** nums = [2,3,-1,8,4]

**Output:** 3

**Explanation:** The sum of the numbers before index 3 is: 2 + 3 + -1 = 4

The sum of the numbers after index 3 is: 4 = 4

My sol:

class Solution:

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

a=sum(nums)

if sum(nums[1:])==0:

return 0

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

nums[x]=nums[x]+nums[x-1]

if nums[x]==a-nums[x-1]:

return x

return -1

**1995. Count Special Quadruplets**

Given a **0-indexed** integer array nums, return *the number of****distinct****quadruplets* (a, b, c, d) *such that:*

* nums[a] + nums[b] + nums[c] == nums[d], and
* a < b < c < d

**Example 1:**

**Input:** nums = [1,2,3,6]

**Output:** 1

**Explanation:** The only quadruplet that satisfies the requirement is (0, 1, 2, 3) because 1 + 2 + 3 == 6.

My sol:

class Solution:

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

p=0

for a in range(0,len(nums)):

for b in range(a+1,len(nums)):

for c in range(b+1,len(nums)):

for d in range(c+1,len(nums)):

if nums[a] + nums[b] + nums[c] == nums[d]:

p+=1

return p

**2000. Reverse Prefix of Word**

Given a **0-indexed** string word and a character ch, **reverse** the segment of word that starts at index 0 and ends at the index of the **first occurrence** of ch (**inclusive**). If the character ch does not exist in word, do nothing.

* For example, if word = "abcdefd" and ch = "d", then you should **reverse** the segment that starts at 0 and ends at 3 (**inclusive**). The resulting string will be "dcbaefd".

Return *the resulting string*.

**Example 1:**

**Input:** word = "abcdefd", ch = "d"

**Output:** "dcbaefd"

**Explanation:** The first occurrence of "d" is at index 3.

Reverse the part of word from 0 to 3 (inclusive), the resulting string is "dcbaefd".

class Solution:

def reversePrefix(self, word: str, ch: str) -> str:

for x in range(0,len(word)):

if word[x]==ch:

word=word.replace(word[:x+1],word[:x+1][::-1])

break

return word;

My sol:

class Solution:

def reversePrefix(self, word: str, ch: str) -> str:

for x in range(0,len(word)):

if word[x]==ch:

word=word.replace(word[:x+1],word[:x+1][::-1])

break

return word;

2001-2050 ($$$$$ 12)

**2006. Count Number of Pairs With Absolute Difference K**

Given an integer array nums and an integer k, return *the number of pairs* (i, j) *where* i < j *such that* |nums[i] - nums[j]| == k.

The value of |x| is defined as:

* x if x >= 0.
* -x if x < 0.

**Example 1:**

**Input:** nums = [1,2,2,1], k = 1

**Output:** 4

**Explanation:** The pairs with an absolute difference of 1 are:

- [**1**,**2**,2,1]

- [**1**,2,**2**,1]

- [1,**2**,2,**1**]

- [1,2,**2**,**1**]

My sol:

class Solution:

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

count=0;

for x in range(0,len(nums)):

for y in range(len(nums)):

if x<y and abs(nums[x] - nums[y]) == k:

count+=1;

return count;

optimized:

count = 0

hash = {}

for i in nums:

if i in hash:

hash[i] +=1

else:

hash[i] = 1

for i in hash:

if i+k in hash:

count+=hash[i]\*hash[i+k]

return count

**2011. Final Value of Variable After Performing Operations**

There is a programming language with only **four** operations and **one** variable X:

* ++X and X++ **increments** the value of the variable X by 1.
* --X and X-- **decrements** the value of the variable X by 1.

Initially, the value of X is 0.

Given an array of strings operations containing a list of operations, return *the****final****value of*X *after performing all the operations*.

**Example 1:**

**Input:** operations = ["--X","X++","X++"]

**Output:** 1

**Explanation:** The operations are performed as follows:

Initially, X = 0.

--X: X is decremented by 1, X = 0 - 1 = -1.

X++: X is incremented by 1, X = -1 + 1 = 0.

X++: X is incremented by 1, X = 0 + 1 = 1.

My sol:

class Solution:

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

X=0;

for ch in operations:

print(ch)

if ch=='--X':

X-=1

elif ch=='X--':

X-=1

elif ch=='X++':

X+=1

elif ch=='++X':

X+=1

else:

pass

return X;

**2016. Maximum Difference Between Increasing Elements**

Given a **0-indexed** integer array nums of size n, find the **maximum difference** between nums[i] and nums[j] (i.e., nums[j] - nums[i]), such that 0 <= i < j < n and nums[i] < nums[j].

Return *the****maximum difference****.*If no such i and j exists, return -1.

**Example 1:**

**Input:** nums = [7,**1**,**5**,4]

**Output:** 4

**Explanation:**

The maximum difference occurs with i = 1 and j = 2, nums[j] - nums[i] = 5 - 1 = 4.

Note that with i = 1 and j = 0, the difference nums[j] - nums[i] = 7 - 1 = 6, but i > j, so it is not valid.

My sol:

class Solution:

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

initial=-1

for x in range(0,len(nums)):

for y in range(0,len(nums)):

if x<y and nums[x] < nums[y] and abs(nums[y]-nums[x]) > initial:

initial= abs(nums[y]-nums[x])

return initial

**2022. Convert 1D Array Into 2D Array**

You are given a **0-indexed** 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with m rows and n columns using **all** the elements from original.

The elements from indices 0 to n - 1 (**inclusive**) of original should form the first row of the constructed 2D array, the elements from indices n to 2 \* n - 1 (**inclusive**) should form the second row of the constructed 2D array, and so on.

Return *an*m x n*2D array constructed according to the above procedure, or an empty 2D array if it is impossible*.

**Example 1:**

A picture containing text, clock

Description automatically generated

**Input:** original = [1,2,3,4], m = 2, n = 2

**Output:** [[1,2],[3,4]]

**Explanation:** The constructed 2D array should contain 2 rows and 2 columns.

The first group of n=2 elements in original, [1,2], becomes the first row in the constructed 2D array.

The second group of n=2 elements in original, [3,4], becomes the second row in the constructed 2D array.

My sol:

class Solution:

def construct2DArray(self, original: List[int], m: int, n: int) -> List[List[int]]:

temp=0

a=[]

b=[]

i=0

if m\*n!=len(original):

return []

for x in range(0,m):

for y in range(0,n):

b.append(original[i])

i+=1

a.append(b)

b=[]

if i==len(original)-1:

return []

return a

**2023. Number of Pairs of Strings With Concatenation Equal to Target**

Given an array of **digit** strings nums and a **digit** string target, return the number of pairs of indices (i, j) (where i != j) such that the ***concatenation*** of nums[i] + nums[j] equals target.

**Example 1:**

**Input:** nums1 = [1,1,3,2], nums2 = [2,3], nums3 = [3]

**Output:** [3,2]

**Explanation:** The values that are present in at least two arrays are:

- 3, in all three arrays.

- 2, in nums1 and nums2.

My sol:

class Solution:

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

a=[]

for x in range(0,len(nums)):

for y in range(0,len(nums)):

if x!=y and nums[x]+nums[y]==target and (x,y) not in a:

a.append((x,y))

return len(a)

**2032. Two Out of Three**

Given three integer arrays nums1, nums2, and nums3, return *a****distinct****array containing all the values that are present in****at least two****out of the three arrays. You may return the values in****any****order*.

**Example 1:**

**Input:** nums = ["777","7","77","77"], target = "7777"

**Output:** 4

**Explanation:** Valid pairs are:

- (0, 1): "777" + "7"

- (1, 0): "7" + "777"

- (2, 3): "77" + "77"

- (3, 2): "77" + "77"

My sol:

class Solution:

def twoOutOfThree(self, nums1: List[int], nums2: List[int], nums3: List[int]) -> List[int]:

s=[]

dict1={}

for x in nums1:

if x in dict1:

pass

else:

dict1[x]=x

dict2={}

for x in nums2:

if x in dict2:

pass

else:

dict2[x]=x

dict3={}

for x in nums3:

if x in dict3:

pass

else:

dict3[x]=x

for key1,value1 in dict1.items():

if key1 in dict2 or key1 in dict3:

s.append(key1)

for key2,value2 in dict2.items():

if key2 in dict3 or key2 in dict1:

s.append(key2)

for key3,value3 in dict3.items():

if key3 in dict1 or key3 in dict2:

s.append(key3)

return list(set(s))

**2037. Minimum Number of Moves to Seat Everyone**

There are n seats and n students in a room. You are given an array seats of length n, where seats[i] is the position of the ith seat. You are also given the array students of length n, where students[j] is the position of the jth student.

You may perform the following move any number of times:

* Increase or decrease the position of the ith student by 1 (i.e., moving the ith student from position x to x + 1 or x - 1)

Return *the****minimum number of moves****required to move each student to a seat such that no two students are in the same seat.*

Note that there may be **multiple** seats or students in the **same**position at the beginning.

**Example 1:**

**Input:** seats = [3,1,5], students = [2,7,4]

**Output:** 4

**Explanation:** The students are moved as follows:

- The first student is moved from from position 2 to position 1 using 1 move.

- The second student is moved from from position 7 to position 5 using 2 moves.

- The third student is moved from from position 4 to position 3 using 1 move.

In total, 1 + 2 + 1 = 4 moves were used.

**My sol:**

class Solution:

def minMovesToSeat(self, seats: List[int], students: List[int]) -> int:

a=sorted(seats)

b=sorted(students)

c=0

for x in range(0,len(students)):

c+=abs(b[x]-a[x])

return c

**2038. Remove Colored Pieces if Both Neighbors are the Same Color**

There are n pieces arranged in a line, and each piece is colored either by 'A' or by 'B'. You are given a string colors of length n where colors[i] is the color of the ith piece.

Alice and Bob are playing a game where they take **alternating turns** removing pieces from the line. In this game, Alice moves**first**.

* Alice is only allowed to remove a piece colored 'A' if **both its neighbors** are also colored 'A'. She is **not allowed** to remove pieces that are colored 'B'.
* Bob is only allowed to remove a piece colored 'B' if **both its neighbors** are also colored 'B'. He is **not allowed** to remove pieces that are colored 'A'.
* Alice and Bob **cannot** remove pieces from the edge of the line.
* If a player cannot make a move on their turn, that player **loses** and the other player **wins**.

Assuming Alice and Bob play optimally, return true*if Alice wins, or return*false*if Bob wins*.

**Example 1:**

**Input:** colors = "AAABABB"

**Output:** true

**Explanation:**

AAABABB -> AABABB

Alice moves first.

She removes the second 'A' from the left since that is the only 'A' whose neighbors are both 'A'.

Now it's Bob's turn.

Bob cannot make a move on his turn since there are no 'B's whose neighbors are both 'B'.

Thus, Alice wins, so return true.

**My sol:**

class Solution:

def winnerOfGame(self, colors: str) -> bool:

countA=0

countB=0

for x in range(1,len(colors)-1):

if colors[x-1] == colors[x] == colors[x+1]:

if colors[x] == 'A':

countA += 1

else:

countB += 1

return countA>countB

**2042. Check if Numbers Are Ascending in a Sentence ( regex for number in sentence)**

A sentence is a list of **tokens** separated by a **single** space with no leading or trailing spaces. Every token is either a **positive number** consisting of digits 0-9 with no leading zeros, or a **word** consisting of lowercase English letters.

* For example, "a puppy has 2 eyes 4 legs" is a sentence with seven tokens: "2" and "4" are numbers and the other tokens such as "puppy" are words.

Given a string s representing a sentence, you need to check if **all** the numbers in s are **strictly increasing** from left to right (i.e., other than the last number, **each** number is **strictly smaller** than the number on its **right** in s).

Return true*if so, or*false*otherwise*.

**Example 1:**

example-1

**Input:** s = "1 box has 3 blue 4 red 6 green and 12 yellow marbles"

**Output:** true

**Explanation:** The numbers in s are: 1, 3, 4, 6, 12.

They are strictly increasing from left to right: 1 < 3 < 4 < 6 < 12.

My sol with regex:

class Solution:

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

flag=0

word = re.findall('(\d+)', s)

nums = [int(i) for i in word]

print(nums)

for x in range(0,len(nums)-1):

if nums[x]<nums[x+1]:

flag=0

else:

flag=1

break

return flag==0

**2046. Sort Linked List Already Sorted Using Absolute Values**

Given the head of a singly linked list that is sorted in **non-decreasing** order using the **absolute values** of its nodes, return the list sorted in ***non-decreasing*** order using the ***actual values*** of its nodes.

**Example 1:**

Diagram

Description automatically generated

**Input:** head = [0,2,-5,5,10,-10]

**Output:** [-10,-5,0,2,5,10]

**Explanation:**

The list sorted in non-descending order using the absolute values of the nodes is [0,2,-5,5,10,-10].

The list sorted in non-descending order using the actual values is [-10,-5,0,2,5,10].

My sol:

class Solution:

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

dummy=ListNode()

p=dummy

cur=head

temp=[]

while cur:

temp.append(cur.val)

cur=cur.next

a=sorted(temp)

for x in a:

p.next=ListNode(x)

p=p.next

return dummy.next

**2047. Number of Valid Words in a Sentence**

A sentence consists of lowercase letters ('a' to 'z'), digits ('0' to '9'), hyphens ('-'), punctuation marks ('!', '.', and ','), and spaces (' ') only. Each sentence can be broken down into **one or more tokens** separated by one or more spaces ' '.

A token is a valid word if **all three** of the following are true:

* It only contains lowercase letters, hyphens, and/or punctuation (**no** digits).
* There is **at most one** hyphen '-'. If present, it **must** be surrounded by lowercase characters ("a-b" is valid, but "-ab" and "ab-" are not valid).
* There is **at most one** punctuation mark. If present, it **must** be at the **end** of the token ("ab,", "cd!", and "." are valid, but "a!b" and "c.," are not valid).

Examples of valid words include "a-b.", "afad", "ba-c", "a!", and "!".

Given a string sentence, return *the****number****of valid words in*sentence.

**Example 1:**

**Input:** sentence = "cat and dog"

**Output:** 3

**Explanation:** The valid words in the sentence are "cat", "and", and "dog".

My sol:

class Solution:

def countValidWords(self, sentence: str) -> int:

a=sentence.split()

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

temp=['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',]

flag=0

count=0

c=0

q=0

for x in a:

for y in range(0,len(x)):

if x[y] in digits:

flag=1

break

if x[y]=='-':

q+=1

if q>1:

flag=1

break

if x[y]=='.':

c+=1

if c>1:

flag=1

break

if x[y]==',':

c+=1

if c>1:

flag=1

break

if x[y]=='!':

c+=1

if c>1:

flag=1

break

if y!=len(x)-1 and (x[y]=='.' or x[y]==',' or x[y]=='!'):

flag=1

break

if x[y] in digits:

flag=1

break

if x[y]=='-':

if y==0 or y==len(x)-1:

flag=1

break

if x[y-1] not in temp or x[y+1] not in temp:

flag=1

break

if flag==1:

count+=0

if flag==0:

count+=1

flag=0

c=0

q=0

return count

2050-2100 ($$$$$ 14)

**2053. Kth Distinct String in an Array**

A **distinct string** is a string that is present only **once** in an array.

Given an array of strings arr, and an integer k, return *the*kth***distinct string****present in*arr. If there are **fewer** than k distinct strings, return *an****empty string***"".

Note that the strings are considered in the **order in which they appear** in the array.

**Example 1:**

**Input:** arr = ["d","b","c","b","c","a"], k = 2

**Output:** "a"

**Explanation:**

The only distinct strings in arr are "d" and "a".

"d" appears 1st, so it is the 1st distinct string.

"a" appears 2nd, so it is the 2nd distinct string.

Since k == 2, "a" is returned.

My sol:

class Solution:

def kthDistinct(self, arr: List[str], k: int) -> str:

a=[]

for x in arr:

if arr.count(x)==1:

a.append(x)

if len(a)<k:

return ""

return a[k-1]

**2057. Smallest Index With Equal Value**

Given a **0-indexed** integer array nums, return *the****smallest****index*i*of*nums*such that*i mod 10 == nums[i]*, or*-1*if such index does not exist*.

x mod y denotes the **remainder** when x is divided by y.

**Example 1:**

**Input:** nums = [0,1,2]

**Output:** 0

**Explanation:**

i=0: 0 mod 10 = 0 == nums[0].

i=1: 1 mod 10 = 1 == nums[1].

i=2: 2 mod 10 = 2 == nums[2].

All indices have i mod 10 == nums[i], so we return the smallest index 0.

My sol:

class Solution:

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

b=[]

for x in range(0,len(nums)):

if x%10==nums[x]:

b.append(x)

print(b)

if len(b)==0:

return -1

return b[0]

**2062. Count Vowel Substrings of a String**

A **substring** is a contiguous (non-empty) sequence of characters within a string.

A **vowel substring** is a substring that **only** consists of vowels ('a', 'e', 'i', 'o', and 'u') and has **all five** vowels present in it.

Given a string word, return *the number of****vowel substrings****in* word.

**Example 1:**

**Input:** word = "aeiouu"

**Output:** 2

**Explanation:** The vowel substrings of word are as follows (underlined):

- "**aeiou**u"

- "**aeiouu**"

My sol:

class Solution:

def countVowelSubstrings(self, word: str) -> int:

i=0

j=0

count=0

while i<=len(word):

while j<=i:

if 'a' in word[j:i] and 'e' in word[j:i] and 'i' in word[j:i] and 'o' in word[j:i] and 'u' in word[j:i] and word[j:i].count('a')+word[j:i].count('e')+word[j:i].count('i')+word[j:i].count('o')+word[j:i].count('u')==len(word[j:i]):

count+=1

j+=1

i+=1

j=0

return count

**2068. Check Whether Two Strings are Almost Equivalent**

Two strings word1 and word2 are considered **almost equivalent** if the differences between the frequencies of each letter from 'a' to 'z' between word1 and word2 is **at most** 3.

Given two strings word1 and word2, each of length n, return true *if*word1 *and* word2 *are****almost equivalent****, or* false *otherwise*.

The **frequency** of a letter x is the number of times it occurs in the string.

**Example 1:**

**Input:** word1 = "aaaa", word2 = "bccb"

**Output:** false

**Explanation:** There are 4 'a's in "aaaa" but 0 'a's in "bccb".

The difference is 4, which is more than the allowed 3.

My sol:

class Solution:

def checkAlmostEquivalent(self, word1: str, word2: str) -> bool:

for x in word1:

if abs(word1.count(x)-word2.count(x))<=3:

flag1=0

else:

flag1=1

break

for x in word2:

if abs(word2.count(x)-word1.count(x))<=3:

flag2=0

else:

flag2=1

break

if flag1==0 and flag2==0:

return True

else:

return False

**2073. Time Needed to Buy Tickets**

There are n people in a line queuing to buy tickets, where the 0th person is at the **front** of the line and the (n - 1)th person is at the **back** of the line.

You are given a **0-indexed** integer array tickets of length n where the number of tickets that the ith person would like to buy is tickets[i].

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to **the end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave**the line.

Return *the****time taken****for the person at position*k***(0-indexed)****to finish buying tickets*.

**Example 1:**

**Input:** tickets = [2,3,2], k = 2

**Output:** 6

**Explanation:**

- In the first pass, everyone in the line buys a ticket and the line becomes [1, 2, 1].

- In the second pass, everyone in the line buys a ticket and the line becomes [0, 1, 0].

The person at position 2 has successfully bought 2 tickets and it took 3 + 3 = 6 seconds.

My sol:

class Solution:

def timeRequiredToBuy(self, tickets: List[int], k: int) -> int:

count=0

x=0

while tickets[k]!=0:

for x in range(0,len(tickets)):

if tickets[x]!=0:

tickets[x]-=1

count+=1

if tickets[k]==0:

break

return count

**2078. Two Furthest Houses With Different Colors**

There are n houses evenly lined up on the street, and each house is beautifully painted. You are given a **0-indexed** integer array colors of length n, where colors[i] represents the color of the ith house.

Return *the****maximum****distance between****two****houses with****different****colors*.

The distance between the ith and jth houses is abs(i - j), where abs(x) is the **absolute value** of x.

**Example 1:**

A picture containing text, clipart

Description automatically generated

**Input:** colors = [**1**,1,1,**6**,1,1,1]

**Output:** 3

**Explanation:** In the above image, color 1 is blue, and color 6 is red.

The furthest two houses with different colors are house 0 and house 3.

House 0 has color 1, and house 3 has color 6. The distance between them is abs(0 - 3) = 3.

Note that houses 3 and 6 can also produce the optimal answer.

My sol:

class Solution:

def maxDistance(self, colors: List[int]) -> int:

maxval=-1

for x in range(0,len(colors)):

for y in range(0,len(colors)):

if x!=y and abs(x-y)>maxval and colors[x]!=colors[y]:

maxval=abs(x-y)

return maxval

**2079. Watering Plants**

You want to water n plants in your garden with a watering can. The plants are arranged in a row and are labeled from 0 to n - 1 from left to right where the ith plant is located at x = i. There is a river at x = -1 that you can refill your watering can at.

Each plant needs a specific amount of water. You will water the plants in the following way:

* Water the plants in order from left to right.
* After watering the current plant, if you do not have enough water to **completely** water the next plant, return to the river to fully refill the watering can.
* You **cannot** refill the watering can early.

You are initially at the river (i.e., x = -1). It takes **one step** to move **one unit** on the x-axis.

Given a **0-indexed** integer array plants of n integers, where plants[i] is the amount of water the ith plant needs, and an integer capacity representing the watering can capacity, return *the****number of steps****needed to water all the plants*.

**Example 1:**

**Input:** plants = [2,2,3,3], capacity = 5

**Output:** 14

**Explanation:** Start at the river with a full watering can:

- Walk to plant 0 (1 step) and water it. Watering can has 3 units of water.

- Walk to plant 1 (1 step) and water it. Watering can has 1 unit of water.

- Since you cannot completely water plant 2, walk back to the river to refill (2 steps).

- Walk to plant 2 (3 steps) and water it. Watering can has 2 units of water.

- Since you cannot completely water plant 3, walk back to the river to refill (3 steps).

- Walk to plant 3 (4 steps) and water it.

Steps needed = 1 + 1 + 2 + 3 + 3 + 4 = 14.

My sol:

class Solution:

def wateringPlants(self, plants: List[int], capacity: int) -> int:

count=0

dummy=capacity

for x in range(0,len(plants)):

if capacity >= plants[x]:

capacity-=plants[x]

count+=1

else:

capacity=dummy

count+=2\*x+1

capacity-=plants[x]

return count

**2083. Substrings That Begin and End With the Same Letter**

You are given a **0-indexed** string s consisting of only lowercase English letters. Return *the number of****substrings****in*s *that begin and end with the****same****character.*

A **substring** is a contiguous non-empty sequence of characters within a string.

**Example 1:**

**Input:** s = "abcba"

**Output:** 7

**Explanation:**

The substrings of length 1 that start and end with the same letter are: "a", "b", "c", "b", and "a".

The substring of length 3 that starts and ends with the same letter is: "bcb".

The substring of length 5 that starts and ends with the same letter is: "abcba".

My sol:

class Solution:

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

dict={}

count=0

for x in s:

if x in dict:

dict[x]+=1

else:

dict[x]=1

count+=dict[x]

return count

**2085. Count Common Words With One Occurrence**

Given two string arrays words1 and words2, return the number of strings that appear ***exactly once*** in **each** of the two arrays.

**Example 1:**

**Input:** words1 = ["leetcode","is","amazing","as","is"], words2 = ["amazing","leetcode","is"]

**Output:** 2

**Explanation:**

- "leetcode" appears exactly once in each of the two arrays. We count this string.

- "amazing" appears exactly once in each of the two arrays. We count this string.

- "is" appears in each of the two arrays, but there are 2 occurrences of it in words1. We do not count this string.

- "as" appears once in words1, but does not appear in words2. We do not count this string.

Thus, there are 2 strings that appear exactly once in each of the two arrays.

**My sol:**

class Solution:

def countWords(self, words1: List[str], words2: List[str]) -> int:

count=0

for x in words1:

if x in words2 and words2.count(x)==1 and words1.count(x)==1:

count+=1

return count

**2089. Find Target Indices After Sorting Array**

You are given a **0-indexed** integer array nums and a target element target.

A **target index** is an index i such that nums[i] == target.

Return *a list of the target indices of* nums after*sorting*nums*in****non-decreasing****order*. If there are no target indices, return *an****empty****list*. The returned list must be sorted in **increasing** order.

**Example 1:**

**Input:** nums = [1,2,5,2,3], target = 2

**Output:** [1,2]

**Explanation:** After sorting, nums is [1,**2**,**2**,3,5].

The indices where nums[i] == 2 are 1 and 2.

My sol

class Solution:

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

a=sorted(nums)

b=[]

for x in range(0,len(a)):

if a[x]==target:

b.append(x)

return b

**2091. Removing Minimum and Maximum From Array**

You are given a **0-indexed** array of **distinct** integers nums.

There is an element in nums that has the **lowest** value and an element that has the **highest** value. We call them the **minimum** and **maximum** respectively. Your goal is to remove **both** these elements from the array.

A **deletion** is defined as either removing an element from the **front** of the array or removing an element from the **back** of the array.

Return *the****minimum****number of deletions it would take to remove****both****the minimum and maximum element from the array.*

**Example 1:**

**Input:** nums = [2,**10**,7,5,4,**1**,8,6]

**Output:** 5

**Explanation:**

The minimum element in the array is nums[5], which is 1.

The maximum element in the array is nums[1], which is 10.

We can remove both the minimum and maximum by removing 2 elements from the front and 3 elements from the back.

This results in 2 + 3 = 5 deletions, which is the minimum number possible.

**My sol:**

class Solution:

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

a=min(nums)

b=max(nums)

i=nums.index(a)

j=nums.index(b)

print(len(nums))

print("minindex : i :",i)

print("maxindex : j :",j)

if len(nums)==1:

return 1

elif i>j:

return min(-abs(j-i)+len(nums)+1,abs(i)+1,len(nums)-j)

elif j>i:

return min(-abs(i-j)+len(nums)+1,abs(j)+1,len(nums)-i)

**2094. Finding 3-Digit Even Numbers**

You are given an integer array digits, where each element is a digit. The array may contain duplicates.

You need to find **all** the **unique** integers that follow the given requirements:

* The integer consists of the **concatenation** of **three** elements from digits in **any** arbitrary order.
* The integer does not have **leading zeros**.
* The integer is **even**.

For example, if the given digits were [1, 2, 3], integers 132 and 312 follow the requirements.

Return *a****sorted****array of the unique integers.*

**Example 1:**

**Input:** digits = [2,1,3,0]

**Output:** [102,120,130,132,210,230,302,310,312,320]

**Explanation:** All the possible integers that follow the requirements are in the output array.

Notice that there are no **odd** integers or integers with **leading zeros**.

My sol:

class Solution:

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

d=collections.Counter(digits)

ans=[]

for x in range(100,1000):

flag=0

if (x%2)==0:

for y in str(x):

if str(x).count(y)<=d[int(y)]:

flag=0

else:

flag=1

break

# print(flag)

if flag==0:

ans.append(int(x))

return ans

**2095. Delete the Middle Node of a Linked List**

You are given the head of a linked list. **Delete** the **middle node**, and return *the* head *of the modified linked list*.

The **middle node** of a linked list of size n is the ⌊n / 2⌋th node from the **start** using **0-based indexing**, where ⌊x⌋ denotes the largest integer less than or equal to x.

* For n = 1, 2, 3, 4, and 5, the middle nodes are 0, 1, 1, 2, and 2, respectively.

**Example 1:**

A picture containing text, pool ball, gambling house

Description automatically generated

**Input:** head = [1,3,4,7,1,2,6]

**Output:** [1,3,4,1,2,6]

**Explanation:**

The above figure represents the given linked list. The indices of the nodes are written below.

Since n = 7, node 3 with value 7 is the middle node, which is marked in red.

We return the new list after removing this node.

My sol:

class Solution:

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

cur=head

a=[]

while cur:

a.append(cur.val)

cur=cur.next

temp=len(a)//2

if temp==0:

return None

count=0

cur=head

while cur:

if count==temp-1:

cur.next=cur.next.next

cur=cur.next

count+=1

return head

**2099. Find Subsequence of Length K With the Largest Sum**

You are given an integer array nums and an integer k. You want to find a **subsequence**of nums of length k that has the **largest** sum.

Return***any****such subsequence as an integer array of length*k.

A **subsequence** is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

**Example 1:**

**Input:** nums = [2,1,3,3], k = 2

**Output:** [3,3]

**Explanation:**

The subsequence has the largest sum of 3 + 3 = 6.

My sol:

class Solution:

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

b=[]

a=sorted(nums)[::-1][:k]

print(a)

for x in nums:

if x in a and b.count(x)!=a.count(x):

b.append(x)

print(b)

return b

2101-2150 ( $$$$$ 17)

**2103. Rings and Rods**

There are n rings and each ring is either red, green, or blue. The rings are distributed **across ten rods** labeled from 0 to 9.

You are given a string rings of length 2n that describes the n rings that are placed onto the rods. Every two characters in rings forms a **color-position pair** that is used to describe each ring where:

* The **first** character of the ith pair denotes the ith ring's **color** ('R', 'G', 'B').
* The **second** character of the ith pair denotes the **rod** that the ith ring is placed on ('0' to '9').

For example, "R3G2B1" describes n == 3 rings: a red ring placed onto the rod labeled 3, a green ring placed onto the rod labeled 2, and a blue ring placed onto the rod labeled 1.

Return *the number of rods that have****all three colors****of rings on them.*

**Example 1:**

A picture containing quoits

Description automatically generated

**Input:** rings = "B0B6G0R6R0R6G9"

**Output:** 1

**Explanation:**

- The rod labeled 0 holds 3 rings with all colors: red, green, and blue.

- The rod labeled 6 holds 3 rings, but it only has red and blue.

- The rod labeled 9 holds only a green ring.

Thus, the number of rods with all three colors is 1.

My sol:

class Solution:

def countPoints(self, rings: str) -> int:

a=[]

count=0

for x in range(0,len(rings),2):

a.append([rings[x],rings[x+1]])

for x in range(0,10):

if rings.count(str(x))>=3 and ['B',str(x)] in a and ['G',str(x)] in a and ['R',str(x)] in a:

count+=1

return count

**2104. Sum of Subarray Ranges**

You are given an integer array nums. The **range** of a subarray of nums is the difference between the largest and smallest element in the subarray.

Return *the****sum of all****subarray ranges of*nums*.*

A subarray is a contiguous **non-empty** sequence of elements within an array.

**Example 1:**

**Input:** nums = [1,2,3]

**Output:** 4

**Explanation:** The 6 subarrays of nums are the following:

[1], range = largest - smallest = 1 - 1 = 0

[2], range = 2 - 2 = 0

[3], range = 3 - 3 = 0

[1,2], range = 2 - 1 = 1

[2,3], range = 3 - 2 = 1

[1,2,3], range = 3 - 1 = 2

So the sum of all ranges is 0 + 0 + 0 + 1 + 1 + 2 = 4.

My sol:

class Solution:

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

res = 0

n = len(nums)

for i in range(0,n):

l,r = nums[i],nums[i]

for j in range(i, n):

l = min(l, nums[j])

r = max(r, nums[j])

res += r - l

return res

**2108. Find First Palindromic String in the Array**

Given an array of strings words, return *the first****palindromic****string in the array*. If there is no such string, return *an****empty string***"".

A string is **palindromic** if it reads the same forward and backward.

**Example 1:**

**Input:** words = ["abc","car","ada","racecar","cool"]

**Output:** "ada"

**Explanation:** The first string that is palindromic is "ada".

Note that "racecar" is also palindromic, but it is not the first.

My sol:

class Solution:

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

for x in words:

if x[::-1]==x:

return x

return ""

**2109. Adding Spaces to a String**

You are given a **0-indexed** string s and a **0-indexed** integer array spaces that describes the indices in the original string where spaces will be added. Each space should be inserted **before** the character at the given index.

* For example, given s = "EnjoyYourCoffee" and spaces = [5, 9], we place spaces before 'Y' and 'C', which are at indices 5 and 9 respectively. Thus, we obtain "Enjoy **Y**our **C**offee".

Return*the modified string****after****the spaces have been added*

**Example 1:**

**Input:** s = "LeetcodeHelpsMeLearn", spaces = [8,13,15]

**Output:** "Leetcode Helps Me Learn"

**Explanation:**

The indices 8, 13, and 15 correspond to the underlined characters in "Leetcode**H**elps**M**e**L**earn".

We then place spaces before those characters.

My sol:

class Solution:

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

i=0

temp=''

for x in range(0,len(spaces)):

temp+=s[i:spaces[x]]

temp+=' '

i=spaces[x]

temp+=s[i:len(s)]

return temp

**2114. Maximum Number of Words Found in Sentences**

A **sentence** is a list of **words** that are separated by a single space with no leading or trailing spaces.

You are given an array of strings sentences, where each sentences[i] represents a single **sentence**.

Return *the****maximum number of words****that appear in a single sentence*.

**Example 1:**

**Input:** sentences = ["alice and bob love leetcode", "i think so too", "this is great thanks very much"]

**Output:** 6

**Explanation:**

- The first sentence, "alice and bob love leetcode", has 5 words in total.

- The second sentence, "i think so too", has 4 words in total.

- The third sentence, "this is great thanks very much", has 6 words in total.

Thus, the maximum number of words in a single sentence comes from the third sentence, which has 6 words.

My sol:

class Solution:

def mostWordsFound(self, sentences: List[str]) -> int:

final=0

for x in sentences:

if len(x.split())>final:

final=len(x.split())

return final

**2119. A Number After a Double Reversal**

**Reversing** an integer means to reverse all its digits.

* For example, reversing 2021 gives 1202. Reversing 12300 gives 321 as the **leading zeros are not retained**.

Given an integer num, **reverse** num to get reversed1, **then reverse** reversed1 to get reversed2. Return true *if* reversed2 *equals* num. Otherwise return false.

**Example 1:**

**Input:** num = 526

**Output:** true

**Explanation:** Reverse num to get 625, then reverse 625 to get 526, which equals num.

My sol:

class Solution:

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

return int(str(int(str(num)[::-1]))[::-1])==num

Other:

class Solution:

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

return (num == 0) or (num % 10)

**2124. Check if All A's Appears Before All B's**

Given a string s consisting of **only** the characters 'a' and 'b', return true if ***every*** 'a' appears before ***every*** 'b' in the string. Otherwise, return false

**Example 1:**

**Input:** s = "aaabbb"

**Output:** true

**Explanation:**

The 'a's are at indices 0, 1, and 2, while the 'b's are at indices 3, 4, and 5.

Hence, every 'a' appears before every 'b' and we return true.

**My sol:**

class Solution:

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

temp1=s.count('a')

for x in s:

if x=='a':

temp1-=1

if x=='b' and temp1>0:

return False

return True

**2125. Number of Laser Beams in a Bank**

Anti-theft security devices are activated inside a bank. You are given a **0-indexed** binary string array bank representing the floor plan of the bank, which is an m x n 2D matrix. bank[i] represents the ith row, consisting of '0's and '1's. '0' means the cell is empty, while'1' means the cell has a security device.

There is **one** laser beam between any **two** security devices **if both** conditions are met:

* The two devices are located on two **different rows**: r1 and r2, where r1 < r2.
* For **each** row i where r1 < i < r2, there are **no security devices** in the ith row.

Laser beams are independent, i.e., one beam does not interfere nor join with another.

Return *the total number of laser beams in the bank*.

**Example 1:**

Chart

Description automatically generated with medium confidence

**Input:** bank = ["011001","000000","010100","001000"]

**Output:** 8

**Explanation:** Between each of the following device pairs, there is one beam. In total, there are 8 beams:

\* bank[0][1] -- bank[2][1]

\* bank[0][1] -- bank[2][3]

\* bank[0][2] -- bank[2][1]

\* bank[0][2] -- bank[2][3]

\* bank[0][5] -- bank[2][1]

\* bank[0][5] -- bank[2][3]

\* bank[2][1] -- bank[3][2]

\* bank[2][3] -- bank[3][2]

Note that there is no beam between any device on the 0th row with any on the 3rd row.

This is because the 2nd row contains security devices, which breaks the second condition.

My sol:

class Solution:

def numberOfBeams(self, bank: List[str]) -> int:

a=[]

for x in bank:

if x.count('1')!=0:

a.append(x)

temp=0

res=0

for x in range(0,len(a)-1):

temp=(a[x].count('1'))\*(a[x+1].count('1'))

res+=temp

temp=0

return res

**2128. Remove All Ones With Row and Column Flips**

You are given an m x n binary matrix grid.

In one operation, you can choose **any** row or column and flip each value in that row or column (i.e., changing all 0's to 1's, and all 1's to 0's).

Return true*if it is possible to remove all*1*'s from*grid using **any** number of operations or false otherwise.

**Example 1:**

A picture containing text, clock

Description automatically generated

**Input:** grid = [[0,1,0],[1,0,1],[0,1,0]]

**Output:** true

**Explanation:** One possible way to remove all 1's from grid is to:

- Flip the middle row

- Flip the middle column

**My sol**

class Solution:

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

pattern=grid[0]

flag=0

temp=[]

final=[]

for x in range(0,len(grid)):

for y in range(0,len(grid[0])):

temp.append(grid[x][y]^pattern[y])

final.append(temp)

temp=[]

print(final)

for x in final:

if len(list(set(x)))!=1:

return False

return True

**2129. Capitalize the Title**

You are given a string title consisting of one or more words separated by a single space, where each word consists of English letters. **Capitalize** the string by changing the capitalization of each word such that:

* If the length of the word is 1 or 2 letters, change all letters to lowercase.
* Otherwise, change the first letter to uppercase and the remaining letters to lowercase.

Return *the****capitalized***title.

**Example 1:**

**Input:** title = "capiTalIze tHe titLe"

**Output:** "Capitalize The Title"

**Explanation:**

Since all the words have a length of at least 3, the first letter of each word is uppercase, and the remaining letters are lowercase.

My sol:

class Solution:

def capitalizeTitle(self, title: str) -> str:

a=title.split()

for x in range(0,len(a)):

if len(a[x])==2 or len(a[x])==1:

a[x]=a[x].lower()

else:

a[x]=a[x][0].upper()+a[x][1:].lower()

return ' '.join(a)

**2130. Maximum Twin Sum of a Linked List**

In a linked list of size n, where n is **even**, the ith node (**0-indexed**) of the linked list is known as the **twin** of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1.

* For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.

The **twin sum**is defined as the sum of a node and its twin.

Given the head of a linked list with even length, return *the****maximum twin sum****of the linked list*.

**Example 1:**

A picture containing text, pool ball, gambling house

Description automatically generated

**Input:** head = [5,4,2,1]

**Output:** 6

**Explanation:**

Nodes 0 and 1 are the twins of nodes 3 and 2, respectively. All have twin sum = 6.

There are no other nodes with twins in the linked list.

Thus, the maximum twin sum of the linked list is 6.

My sol:

class Solution:

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

cur=head

temp=[]

while cur:

temp.append(cur.val)

cur=cur.next

print(temp)

a=len(temp)

i=0

j=a-1

res=0

while i<j:

res=max(res,temp[i]+temp[j])

i+=1

j-=1

return res

**2133. Check if Every Row and Column Contains All Numbers**

An n x n matrix is **valid** if every row and every column contains **all** the integers from 1 to n (**inclusive**).

Given an n x n integer matrix matrix, return true *if the matrix is****valid****.* Otherwise, return false.

**Example 1:**

A picture containing crossword puzzle, shoji

Description automatically generated

**Input:** matrix = [[1,2,3],[3,1,2],[2,3,1]]

**Output:** true

**Explanation:** In this case, n = 3, and every row and column contains the numbers 1, 2, and 3.

Hence, we return true.

My sol:

class Solution:

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

temp1=[]

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

temp1.append(i)

for x in matrix:

if sorted(x)!=sorted(temp1):

return False

j=0

temp=[]

a=[]

while j!=len(matrix[0]):

temp=[]

for x in range(0,len(matrix)):

temp.append(matrix[x][j])

j+=1

a.append(temp)

for x in a:

if sorted(x)!=sorted(temp1):

return False

return True

**2138. Divide a String Into Groups of Size k**

A string s can be partitioned into groups of size k using the following procedure:

* The first group consists of the first k characters of the string, the second group consists of the next k characters of the string, and so on. Each character can be a part of **exactly one** group.
* For the last group, if the string **does not** have k characters remaining, a character fill is used to complete the group.

Note that the partition is done so that after removing the fill character from the last group (if it exists) and concatenating all the groups in order, the resultant string should be s.

Given the string s, the size of each group k and the character fill, return *a string array denoting the****composition of every group***s*has been divided into, using the above procedure*.

**Example 1:**

**Input:** s = "abcdefghi", k = 3, fill = "x"

**Output:** ["abc","def","ghi"]

**Explanation:**

The first 3 characters "abc" form the first group.

The next 3 characters "def" form the second group.

The last 3 characters "ghi" form the third group.

Since all groups can be completely filled by characters from the string, we do not need to use fill.

Thus, the groups formed are "abc", "def", and "ghi".

My sol:

class Solution:

def divideString(self, s: str, k: int, fill: str) -> List[str]:

temp=[]

i=0

while i+k<=len(s):

temp.append(s[i:i+k])

i+=k

res=s[i:]

while len(res)<k and len(res)>0:

res+=fill

print(res)

if res!="":

temp.append(res)

return temp

**2144. Minimum Cost of Buying Candies With Discount**

A shop is selling candies at a discount. For **every two** candies sold, the shop gives a **third** candy for **free**.

The customer can choose **any** candy to take away for free as long as the cost of the chosen candy is less than or equal to the **minimum** cost of the two candies bought.

* For example, if there are 4 candies with costs 1, 2, 3, and 4, and the customer buys candies with costs 2 and 3, they can take the candy with cost 1 for free, but not the candy with cost 4.

Given a **0-indexed** integer array cost, where cost[i] denotes the cost of the ith candy, return *the****minimum cost****of buying****all****the candies*.

**Example 1:**

**Input:** cost = [1,2,3]

**Output:** 5

**Explanation:** We buy the candies with costs 2 and 3, and take the candy with cost 1 for free.

The total cost of buying all candies is 2 + 3 = 5. This is the **only** way we can buy the candies.

Note that we cannot buy candies with costs 1 and 3, and then take the candy with cost 2 for free.

The cost of the free candy has to be less than or equal to the minimum cost of the purchased candies.

My sol:

class Solution:

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

i=0

a=sorted(cost)[::-1]

k=3

res=0

while i+k<=len(cost):

res+=a[i:i+k][0]+a[i:i+k][1]

i+=3

if len(a[i:])==2:

res+=a[i:][0]+a[i:][1]

elif len(a[i:])==1:

res+=a[i:][0]

return res

**2148. Count Elements With Strictly Smaller and Greater Elements**

Given an integer array nums, return the number of elements that have ***both*** a strictly smaller and a strictly greater element appear in nums.

**Example 1:**

**Input:** nums = [11,7,2,15]

**Output:** 2

**Explanation:** The element 7 has the element 2 strictly smaller than it and the element 11 strictly greater than it.

Element 11 has element 7 strictly smaller than it and element 15 strictly greater than it.

In total there are 2 elements having both a strictly smaller and a strictly greater element appear in nums.

My sol:

class Solution:

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

d=collections.Counter(nums)

m=max(nums)

n=min(nums)

d[m]=0

d[n]=0

count=0

for k,v in d.items():

count+=v

return count

**2149. Rearrange Array Elements by Sign**

You are given a **0-indexed** integer array nums of **even** length consisting of an **equal** number of positive and negative integers.

You should **rearrange** the elements of nums such that the modified array follows the given conditions:

1. Every **consecutive pair** of integers have **opposite signs**.
2. For all integers with the same sign, the **order** in which they were present in nums is **preserved**.
3. The rearranged array begins with a positive integer.

Return *the modified array after rearranging the elements to satisfy the aforementioned conditions*.

**Example 1:**

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

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

**Explanation:**

The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4].

The only possible way to rearrange them such that they satisfy all conditions is [3,-2,1,-5,2,-4].

Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions.

My sol:

class Solution:

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

temp1=[]

temp2=[]

for x in nums:

if x<0:

temp1.append(x)

else:

temp2.append(x)

i=0

j=0

res=[]

while i<len(temp1):

res.append(temp2[i])

res.append(temp1[i])

i+=1

return res

**2150. Find All Lonely Numbers in the Array**

You are given an integer array nums. A number x is **lonely** when it appears only **once**, and no **adjacent** numbers (i.e. x + 1 and x - 1) appear in the array.

Return ***all****lonely numbers in*nums. You may return the answer in **any order**.

**Example 1:**

**Input:** nums = [10,6,5,8]

**Output:** [10,8]

**Explanation:**

- 10 is a lonely number since it appears exactly once and 9 and 11 does not appear in nums.

- 8 is a lonely number since it appears exactly once and 7 and 9 does not appear in nums.

- 5 is not a lonely number since 6 appears in nums and vice versa.

Hence, the lonely numbers in nums are [10, 8].

Note that [8, 10] may also be returned.

My soL:

class Solution:

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

a=collections.Counter(nums)

res=[]

for x in nums:

if a[x]==1 and x+1 not in a and x-1 not in a:

res.append(x)

return res

2151-2200 ( $$$$$ 16)

**2154. Keep Multiplying Found Values by Two**

You are given an array of integers nums. You are also given an integer original which is the first number that needs to be searched for in nums.

You then do the following steps:

1. If original is found in nums, **multiply** it by two (i.e., set original = 2 \* original).
2. Otherwise, **stop** the process.
3. **Repeat** this process with the new number as long as you keep finding the number.

Return *the****final****value of*original.

**Example 1:**

**Input:** nums = [5,3,6,1,12], original = 3

**Output:** 24

**Explanation:**

- 3 is found in nums. 3 is multiplied by 2 to obtain 6.

- 6 is found in nums. 6 is multiplied by 2 to obtain 12.

- 12 is found in nums. 12 is multiplied by 2 to obtain 24.

- 24 is not found in nums. Thus, 24 is returned.

**My sol:**

class Solution:

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

while original in nums:

original =original \*2

return original

**2160. Minimum Sum of Four Digit Number After Splitting Digits**

You are given a **positive** integer num consisting of exactly four digits. Split num into two new integers new1 and new2 by using the **digits** found in num. **Leading zeros** are allowed in new1 and new2, and **all** the digits found in num must be used.

* For example, given num = 2932, you have the following digits: two 2's, one 9 and one 3. Some of the possible pairs [new1, new2] are [22, 93], [23, 92], [223, 9] and [2, 329].

Return *the****minimum****possible sum of*new1*and*new2.

**Example 1:**

**Input:** num = 2932

**Output:** 52

**Explanation:** Some possible pairs [new1, new2] are [29, 23], [223, 9], etc.

The minimum sum can be obtained by the pair [29, 23]: 29 + 23 = 52.

My sol:

class Solution:

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

a=sorted(str(num))

return int(a[0]+a[2])+int(a[1]+a[3])

**2161. Partition Array According to Given Pivot**

You are given a **0-indexed** integer array nums and an integer pivot. Rearrange nums such that the following conditions are satisfied:

* Every element less than pivot appears **before** every element greater than pivot.
* Every element equal to pivot appears **in between** the elements less than and greater than pivot.
* The **relative order** of the elements less than pivot and the elements greater than pivot is maintained.
  + More formally, consider every pi, pj where pi is the new position of the ith element and pj is the new position of the jth element. For elements less than pivot, if i < j and nums[i] < pivot and nums[j] < pivot, then pi < pj. Similarly for elements greater than pivot, if i < j and nums[i] > pivot and nums[j] > pivot, then pi < pj.

Return nums*after the rearrangement.*

**Example 1:**

**Input:** nums = [9,12,5,10,14,3,10], pivot = 10

**Output:** [9,5,3,10,10,12,14]

**Explanation:**

The elements 9, 5, and 3 are less than the pivot so they are on the left side of the array.

The elements 12 and 14 are greater than the pivot so they are on the right side of the array.

The relative ordering of the elements less than and greater than pivot is also maintained. [9, 5, 3] and [12, 14] are the respective orderings.

My sol:

class Solution:

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

temp1=[]

temp2=[]

for x in nums:

if x < pivot:

temp1.append(x)

elif x > pivot:

temp2.append(x)

else:

pass

res=[]

for x in range(0,nums.count(pivot)):

res.append(pivot)

return temp1+res+temp2

**2164. Sort Even and Odd Indices Independently**

You are given a **0-indexed** integer array nums. Rearrange the values of nums according to the following rules:

1. Sort the values at **odd indices** of nums in **non-increasing** order.
   * For example, if nums = [4,**1**,2,**3**] before this step, it becomes [4,**3**,2,**1**] after. The values at odd indices 1 and 3 are sorted in non-increasing order.
2. Sort the values at **even indices** of nums in **non-decreasing** order.
   * For example, if nums = [**4**,1,**2**,3] before this step, it becomes [**2**,1,**4**,3] after. The values at even indices 0 and 2 are sorted in non-decreasing order.

Return *the array formed after rearranging the values of* nums.

**Example 1:**

**Input:** nums = [4,1,2,3]

**Output:** [2,3,4,1]

**Explanation:**

First, we sort the values present at odd indices (1 and 3) in non-increasing order.

So, nums changes from [4,**1**,2,**3**] to [4,**3**,2,**1**].

Next, we sort the values present at even indices (0 and 2) in non-decreasing order.

So, nums changes from [**4**,1,**2**,3] to [**2**,3,**4**,1].

Thus, the array formed after rearranging the values is [2,3,4,1].

My sol:

class Solution:

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

temp1=[]

temp2=[]

for x in range(0,len(nums)):

if x%2==0:

temp1.append(nums[x])

else:

temp2.append(nums[x])

a=sorted(temp1)

b=sorted(temp2)[::-1]

res=[]

i=0

j=0

while i<len(a) or j<len(b):

if i<len(a):

res.append(a[i])

if j<len(b):

res.append(b[j])

i+=1

j+=1

return res

**2165. Smallest Value of the Rearranged Number**

You are given an integer num. **Rearrange** the digits of num such that its value is **minimized** and it does not contain **any** leading zeros.

Return *the rearranged number with minimal value*.

Note that the sign of the number does not change after rearranging the digits.

**Example 1:**

**Input:** num = 310

**Output:** 103

**Explanation:** The possible arrangements for the digits of 310 are 013, 031, 103, 130, 301, 310.

The arrangement with the smallest value that does not contain any leading zeros is 103.

My sol:

class Solution:

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

a=sorted(str(abs(num)))

if num==0:

return 0

if num<0:

flag=0

else:

flag=1

if flag==0:

return int(''.join(a[::-1]))\*-1

temp=''

if flag==1 and a[0]=='0':

for x in range(0,len(a)):

if a[x]!='0':

temp=a[x]

b=x

break

return int(temp+''.join(a[:b])+''.join(a[b+1:]))

else:

return ''.join(a)

**2169. Count Operations to Obtain Zero**

You are given two **non-negative** integers num1 and num2.

In one **operation**, if num1 >= num2, you must subtract num2 from num1, otherwise subtract num1 from num2.

* For example, if num1 = 5 and num2 = 4, subtract num2 from num1, thus obtaining num1 = 1 and num2 = 4. However, if num1 = 4 and num2 = 5, after one operation, num1 = 4 and num2 = 1.

Return *the****number of operations****required to make either* num1 = 0 *or* num2 = 0.

**Example 1:**

**Input:** num1 = 2, num2 = 3

**Output:** 3

**Explanation:**

- Operation 1: num1 = 2, num2 = 3. Since num1 < num2, we subtract num1 from num2 and get num1 = 2, num2 = 3 - 2 = 1.

- Operation 2: num1 = 2, num2 = 1. Since num1 > num2, we subtract num2 from num1.

- Operation 3: num1 = 1, num2 = 1. Since num1 == num2, we subtract num2 from num1.

Now num1 = 0 and num2 = 1. Since num1 == 0, we do not need to perform any further operations.

So the total number of operations required is 3.

**My sol:**

class Solution:

def countOperations(self, num1: int, num2: int) -> int:

count=0

while num1!=0 or num2!=0:

if num1==0 or num2==0:

return count

if num1>=num2:

num1=num1-num2

count+=1

else:

num2=num2-num1

count+=1

return count

**2176. Count Equal and Divisible Pairs in an Array**

Given a **0-indexed** integer array nums of length n and an integer k, return the ***number of pairs*** (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i \* j) is divisible by k.

**Example 1:**

**Input:** nums = [3,1,2,2,2,1,3], k = 2

**Output:** 4

**Explanation:**

There are 4 pairs that meet all the requirements:

- nums[0] == nums[6], and 0 \* 6 == 0, which is divisible by 2.

- nums[2] == nums[3], and 2 \* 3 == 6, which is divisible by 2.

- nums[2] == nums[4], and 2 \* 4 == 8, which is divisible by 2.

- nums[3] == nums[4], and 3 \* 4 == 12, which is divisible by 2.

MYSOL:

class Solution:

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

count=0

for x in range(0,len(nums)):

for y in range(x,len(nums)):

if nums[x]==nums[y] and (x\*y)%k==0 and x!=y:

count+=1

return count

**2177. Find Three Consecutive Integers That Sum to a Given Number**

Given an integer num, return three consecutive integers (as a sorted array) that ***sum*** to num. If num cannot be expressed as the sum of three consecutive integers, return an ***empty*** array.

**Example 1:**

**Input:** num = 33

**Output:** [10,11,12]

**Explanation:** 33 can be expressed as 10 + 11 + 12 = 33.

10, 11, 12 are 3 consecutive integers, so we return [10, 11, 12].

**MY SOL :**

class Solution:

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

if num%3!=0:

return []

else:

temp=num//3

return [temp-1,temp,temp+1]

**2180. Count Integers With Even Digit Sum**

Given a positive integer num, return the number of positive integers ***less than or equal to*** num whose digit sums are ***even***.

The **digit sum** of a positive integer is the sum of all its digits.

**Example 1:**

**Input:** num = 4

**Output:** 2

**Explanation:**

The only integers less than or equal to 4 whose digit sums are even are 2 and 4.

**MY SOL:**

class Solution:

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

c=0

count=0

for x in range(1,num+1):

for y in str(x):

c+=int(y)

# print(c)

if c%2==0:

count+=1

c=0

return count

**2181. Merge Nodes in Between Zeros**

You are given the head of a linked list, which contains a series of integers **separated** by 0's. The **beginning** and **end** of the linked list will have Node.val == 0.

For **every**two consecutive 0's, **merge** all the nodes lying in between them into a single node whose value is the **sum** of all the merged nodes. The modified list should not contain any 0's.

Return the head of the modified linked list.

**Example 1:**



**Input:** head = [0,3,1,0,4,5,2,0]

**Output:** [4,11]

**Explanation:**

The above figure represents the given linked list. The modified list contains

- The sum of the nodes marked in green: 3 + 1 = 4.

- The sum of the nodes marked in red: 4 + 5 + 2 = 11.

My sol:

class Solution:

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

dummy=ListNode()

p=dummy

cur=head

temp=[]

while cur:

temp.append(cur.val)

cur=cur.next

k=0

res=[]

final=[]

for x in temp:

if x==0:

final.append(sum(res))

res=[]

else:

res.append(x)

for x in final[1:]:

p.next=ListNode(x)

p=p.next

return dummy.next

**2185. Counting Words With a Given Prefix**

You are given an array of strings words and a string pref.

Return the number of strings in words that contain pref as a ***prefix***.

A **prefix** of a string s is any leading contiguous substring of s.

**Example 1:**

**Input:** words = ["pay","**at**tention","practice","**at**tend"], pref = "at"

**Output:** 2

**Explanation:** The 2 strings that contain "at" as a prefix are: "**at**tention" and "**at**tend".

My sol:

class Solution:

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

count=0

for x in words:

if x[0:len(pref)]==pref:

count+=1

return count

**2190. Most Frequent Number Following Key In an Array**

You are given a **0-indexed** integer array nums.You are also given an integer key, which is present in nums.

For every unique integer target in nums, **count** the number of times target immediately follows an occurrence of key in nums. In other words, count the number of indices i such that:

* 0 <= i <= nums.length - 2,
* nums[i] == key and,
* nums[i + 1] == target.

Return *the*target*with the****maximum****count*. The test cases will be generated such that the target with maximum count is unique.

**Example 1:**

**Input:** nums = [1,100,200,1,100], key = 1

**Output:** 100

**Explanation:** For target = 100, there are 2 occurrences at indices 1 and 4 which follow an occurrence of key.

No other integers follow an occurrence of key, so we return 100.

**My sol:**

class Solution:

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

dict={}

for x in range(0,len(nums)-1):

if nums[x]==key:

if nums[x+1] in dict:

dict[nums[x+1]]+=1

else:

dict[nums[x+1]]=1

mval=0

res=0

for x,y in dict.items():

if y>mval:

mval=y

res=x

return res

**2194. Cells in a Range on an Excel Sheet**

A cell (r, c) of an excel sheet is represented as a string "<col><row>" where:

* <col> denotes the column number c of the cell. It is represented by **alphabetical letters**.
  + For example, the 1st column is denoted by 'A', the 2nd by 'B', the 3rd by 'C', and so on.
* <row> is the row number r of the cell. The rth row is represented by the **integer** r.

You are given a string s in the format "<col1><row1>:<col2><row2>", where <col1> represents the column c1, <row1> represents the row r1, <col2> represents the column c2, and <row2> represents the row r2, such that r1 <= r2 and c1 <= c2.

Return *the****list of cells*** (x, y) *such that* r1 <= x <= r2 *and* c1 <= y <= c2. The cells should be represented as **strings** in the format mentioned above and be sorted in **non-decreasing** order first by columns and then by rows.

**Example 1:**

Chart

Description automatically generated with medium confidence

**Input:** s = "K1:L2"

**Output:** ["K1","K2","L1","L2"]

**Explanation:**

The above diagram shows the cells which should be present in the list.

The red arrows denote the order in which the cells should be presented.

My sol:

class Solution:

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

d={1:'A',2:'B',3:'C',4:'D',5:'E',6:'F',7:'G',8:'H',9:'I',10:'J',11:'K',12:'L',13:'M',14:'N',15:'O',16:'P',17:'Q',18:'R',19:'S',20:'T',21:'U',22:'V',23:'W',24:'X',25:'Y',26:'Z'}

p={'A':1,'B':2,'C':3,'D':4,'E':5,'F':6,'G':7,'H':8,'I':9,'J':10,'K':11,'L':12,'M':13,'N':14,'O':15,'P':16,'Q':17,'R':18,'S':19,'T':20,'U':21,'V':22,'W':23,'X':24,'Y':25,'Z':26}

a=s.split(':')

col1=a[0][0]

col2=a[1][0]

row1=a[0][1]

row2=a[1][1]

res=[]

for i in range(p[col1],p[col2]+1):

for j in range(int(row1),int(row2)+1):

res.append(str(d[i])+str(j))

return res

**2200. Find All K-Distant Indices in an Array**

You are given a **0-indexed** integer array nums and two integers key and k. A **k-distant index** is an index i of nums for which there exists at least one index j such that |i - j| <= k and nums[j] == key.

Return a list of all k-distant indices sorted in ***increasing order***.

**Example 1:**

**Input:** nums = [3,4,9,1,3,9,5], key = 9, k = 1

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

**Explanation:** Here, nums[2] == key and nums[5] == key.

- For index 0, |0 - 2| > k and |0 - 5| > k, so there is no j where |0 - j| <= k and nums[j] == key. Thus, 0 is not a k-distant index.

- For index 1, |1 - 2| <= k and nums[2] == key, so 1 is a k-distant index.

- For index 2, |2 - 2| <= k and nums[2] == key, so 2 is a k-distant index.

- For index 3, |3 - 2| <= k and nums[2] == key, so 3 is a k-distant index.

- For index 4, |4 - 5| <= k and nums[5] == key, so 4 is a k-distant index.

- For index 5, |5 - 5| <= k and nums[5] == key, so 5 is a k-distant index.

- For index 6, |6 - 5| <= k and nums[5] == key, so 6 is a k-distant index.

Thus, we return [1,2,3,4,5,6] which is sorted in increasing order.

My sol:

class Solution:

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

temp=[]

res=[]

for x in range(0,len(nums)):

if nums[x]==key:

temp.append(x)

for x in range(0,len(nums)):

for y in temp:

if abs(x-y)<=k:

res.append(x)

break

return res

2201-2250( $$$$$ 14)

**2206. Divide Array Into Equal Pairs**

You are given an integer array nums consisting of 2 \* n integers.

You need to divide nums into n pairs such that:

* Each element belongs to **exactly one** pair.
* The elements present in a pair are **equal**.

Return true *if nums can be divided into* n *pairs, otherwise return* false.

**Example 1:**

**Input:** nums = [3,2,3,2,2,2]

**Output:** true

**Explanation:**

There are 6 elements in nums, so they should be divided into 6 / 2 = 3 pairs.

If nums is divided into the pairs (2, 2), (3, 3), and (2, 2), it will satisfy all the conditions.

My sol:

class Solution:

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

d=collections.Counter(nums)

for x,y in d.items():

if y%2!=0:

return False

return True

**2210. Count Hills and Valleys in an Array**

You are given a **0-indexed** integer array nums. An index i is part of a **hill** in nums if the closest non-equal neighbors of i are smaller than nums[i]. Similarly, an index i is part of a **valley** in nums if the closest non-equal neighbors of i are larger than nums[i]. Adjacent indices i and j are part of the **same** hill or valley if nums[i] == nums[j].

Note that for an index to be part of a hill or valley, it must have a non-equal neighbor on **both** the left and right of the index.

Return *the number of hills and valleys in*nums.

**Example 1:**

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

**Output:** 3

**Explanation:**

At index 0: There is no non-equal neighbor of 2 on the left, so index 0 is neither a hill nor a valley.

At index 1: The closest non-equal neighbors of 4 are 2 and 1. Since 4 > 2 and 4 > 1, index 1 is a hill.

At index 2: The closest non-equal neighbors of 1 are 4 and 6. Since 1 < 4 and 1 < 6, index 2 is a valley.

At index 3: The closest non-equal neighbors of 1 are 4 and 6. Since 1 < 4 and 1 < 6, index 3 is a valley, but note that it is part of the same valley as index 2.

At index 4: The closest non-equal neighbors of 6 are 1 and 5. Since 6 > 1 and 6 > 5, index 4 is a hill.

At index 5: There is no non-equal neighbor of 5 on the right, so index 5 is neither a hill nor a valley.

There are 3 hills and valleys so we return 3.

**My sol:**

class Solution:

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

count=0

found=0

hill1=0

valley1=0

hill2=0

valley2=0

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

hill1=0

valley1=0

hill2=0

valley2=0

i=x-1

j=x+1

while i>=0:

if nums[x]>nums[i]:

# found=1

hill1=1

break

if nums[x]<nums[i]:

# found=1

valley1=1

break

i-=1

while j<=len(nums)-1:

if nums[x]>nums[j]:

# found=1

hill2=1

break

if nums[x]<nums[j]:

# found=1

valley2=1

break

j+=1

print(hill1,hill2,valley1,valley2)

if hill1==1 and hill2==1 and nums[x+1]!=nums[x]:

print("hill:"+str(nums[x]))

count+=1

if valley1==1 and valley2==1 and nums[x+1]!=nums[x]:

print("valley:"+str(nums[x]))

count+=1

return count

**2215. Find the Difference of Two Arrays**

Given two **0-indexed** integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:*

* answer[0] *is a list of all****distinct****integers in* nums1 *which are****not****present in* nums2*.*
* answer[1] *is a list of all****distinct****integers in* nums2 *which are****not****present in* nums1.

**Note** that the integers in the lists may be returned in **any** order.

**Example 1:**

**Input:** nums1 = [1,2,3], nums2 = [2,4,6]

**Output:** [[1,3],[4,6]]

**Explanation:**

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6]

class Solution:

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

d1=collections.Counter(nums1)

d2=collections.Counter(nums2)

d3={}

d4={}

temp1=[]

temp2=[]

for x in nums1:

if x not in d2 and x not in d3:

d3[x]=x

temp1.append(x)

for x in nums2:

if x not in d1 and x not in d4:

temp2.append(x)

d4[x]=x

res=[temp1,temp2]

return res

**other soL;**

return [set(nums1)-set(nums2), set(nums2)-set(nums1)]

**2219. Maximum Sum Score of Array(kiran Amazon question)(prefix sum one pass left and right)**

You are given a **0-indexed** integer array nums of length n.

The **sum score** of nums at an index i where 0 <= i < n is the **maximum** of:

* The sum of the **first** i + 1 elements of nums.
* The sum of the **last** n - i elements of nums.

Return *the****maximum******sum score****of*nums*at any index.*

**Example 1:**

**Input:** nums = [4,3,-2,5]

**Output:** 10

**Explanation:**

The sum score at index 0 is max(4, 4 + 3 + -2 + 5) = max(4, 10) = 10.

The sum score at index 1 is max(4 + 3, 3 + -2 + 5) = max(7, 6) = 7.

The sum score at index 2 is max(4 + 3 + -2, -2 + 5) = max(5, 3) = 5.

The sum score at index 3 is max(4 + 3 + -2 + 5, 5) = max(10, 5) = 10.

The maximum sum score of nums is 10.

**MY sol:**

class Solution:

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

l=[nums[0]]

r=[sum(nums)]

m=-100000

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

l.append(nums[x]+l[x-1])

r.append(r[x-1]-nums[x-1])

for x in range(0,len(l)):

p=(max(l[x],r[x]))

m=max(m,p)

return m

**2221. Find Triangular Sum of an Array**

You are given a **0-indexed** integer array nums, where nums[i] is a digit between 0 and 9 (**inclusive**).

The **triangular sum** of nums is the value of the only element present in nums after the following process terminates:

1. Let nums comprise of n elements. If n == 1, **end** the process. Otherwise, **create** a new **0-indexed** integer array newNums of length n - 1.
2. For each index i, where 0 <= i < n - 1, **assign** the value of newNums[i] as (nums[i] + nums[i+1]) % 10, where % denotes modulo operator.
3. **Replace** the array nums with newNums.
4. **Repeat** the entire process starting from step 1.

Return *the triangular sum of* nums.

**Example 1:**

A picture containing text, electronics, black, keyboard

Description automatically generated

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

**Output:** 8

**Explanation:**

The above diagram depicts the process from which we obtain the triangular sum of the array.

**My sol:**

class Solution:

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

i=0

temp=[]

while len(nums)!=1:

for x in range(0,len(nums)-1):

temp.append((nums[x]+nums[x+1])%10)

i+=1

nums=temp

temp=[]

i=0

return nums[0]

**2224. Minimum Number of Operations to Convert Time**

You are given two strings current and correct representing two **24-hour times**.

24-hour times are formatted as "HH:MM", where HH is between 00 and 23, and MM is between 00 and 59. The earliest 24-hour time is 00:00, and the latest is 23:59.

In one operation you can increase the time current by 1, 5, 15, or 60 minutes. You can perform this operation **any** number of times.

Return the ***minimum number of operations*** needed to convert current to correct.

**Example 1:**

**Input:** current = "02:30", correct = "04:35"

**Output:** 3

**Explanation:**

We can convert current to correct in 3 operations as follows:

- Add 60 minutes to current. current becomes "03:30".

- Add 60 minutes to current. current becomes "04:30".

- Add 5 minutes to current. current becomes "04:35".

It can be proven that it is not possible to convert current to correct in fewer than 3 operations.

**My sol:**

class Solution:

def convertTime(self, current: str, correct: str) -> int:

a=current.split(':')

b=correct.split(':')

c=int(a[0])\*60 + int(a[1])

d=int(b[0])\*60 + int(b[1])

temp=d-c

arr=[60,15,5,1]

count=0

x=0

while temp>=0:

if temp-arr[x]>=0:

temp-=arr[x]

count+=1

else:

x+=1

if temp==0:

break

return count

**2225. Find Players With Zero or One Losses**

You are given an integer array matches where matches[i] = [winneri, loseri] indicates that the player winneri defeated player loseri in a match.

Return *a list*answer*of size*2*where:*

* answer[0] is a list of all players that have **not** lost any matches.
* answer[1] is a list of all players that have lost exactly **one** match.

The values in the two lists should be returned in **increasing** order.

**Note:**

* You should only consider the players that have played **at least one** match.
* The testcases will be generated such that **no** two matches will have the **same** outcome.

**Example 1:**

**Input:** matches = [[1,3],[2,3],[3,6],[5,6],[5,7],[4,5],[4,8],[4,9],[10,4],[10,9]]

**Output:** [[1,2,10],[4,5,7,8]]

**Explanation:**

Players 1, 2, and 10 have not lost any matches.

Players 4, 5, 7, and 8 each have lost one match.

Players 3, 6, and 9 each have lost two matches.

Thus, answer[0] = [1,2,10] and answer[1] = [4,5,7,8].

**My sol:**

class Solution:

def findWinners(self, matches: List[List[int]]) -> List[List[int]]:

temp1=[]

temp2=[]

final=[]

for x in matches:

temp1.append(x[0])

temp2.append(x[1])

final.append(sorted(list(set(temp1)-set(temp2))))

d=collections.Counter(temp2)

res=[]

for x,y in d.items():

if y==1:

res.append(x)

final.append(sorted(res))

return final

**2229. Check if an Array Is Consecutive**

Given an integer array nums, return true if nums is ***consecutive***, otherwise return false.

An array is **consecutive**if it contains every number in the range [x, x + n - 1] (**inclusive**), where x is the minimum number in the array and n is the length of the array.

**Example 1:**

**Input:** nums = [1,3,4,2]

**Output:** true

**Explanation:**

The minimum value is 1 and the length of nums is 4.

All of the values in the range [x, x + n - 1] = [1, 1 + 4 - 1] = [1, 4] = (1, 2, 3, 4) occur in nums.

Therefore, nums is consecutive.

**My sol:**

class Solution:

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

a=collections.Counter(nums)

for x in range(min(nums),min(nums)+len(nums)):

if x not in a:

return False

return True

**2231. Largest Number After Digit Swaps by Parity**

You are given a positive integer num. You may swap any two digits of num that have the same **parity** (i.e. both odd digits or both even digits).

Return*the****largest****possible value of*num*after****any****number of swaps.*

**Example 1:**

**Input:** num = 1234

**Output:** 3412

**Explanation:** Swap the digit 3 with the digit 1, this results in the number 3214.

Swap the digit 2 with the digit 4, this results in the number 3412.

Note that there may be other sequences of swaps but it can be shown that 3412 is the largest possible number.

Also note that we may not swap the digit 4 with the digit 1 since they are of different parities.

My sol:

class Solution:

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

n=str(num)

odd=[]

even=[]

for x in range(0,len(n)):

if int(n[x]) %2==0:

even.append(int(n[x]))

else:

odd.append(int(n[x]))

e=sorted(even)[::-1]

o=sorted(odd)[::-1]

s=""

i=0

j=0

for x in str(num):

if int(x)%2==0:

s+=str(e[i])

i+=1

else:

s+=str(o[j])

j+=1

print(s)

return s

**2235. Add Two Integers**

Given two integers num1 and num2, return the ***sum*** of the two integers.

**Example 1:**

**Input:** num1 = 12, num2 = 5

**Output:** 17

**Explanation:** num1 is 12, num2 is 5, and their sum is 12 + 5 = 17, so 17 is returned.

My sol:

class Solution:

def sum(self, num1: int, num2: int) -> int:

return num1+num2

**2236. Root Equals Sum of Children**

You are given the root of a **binary tree** that consists of exactly 3 nodes: the root, its left child, and its right child.

Return true *if the value of the root is equal to the****sum****of the values of its two children, or*false*otherwise*.

**Example 1:**

Diagram

Description automatically generated

**Input:** root = [10,4,6]

**Output:** true

**Explanation:** The values of the root, its left child, and its right child are 10, 4, and 6, respectively.

10 is equal to 4 + 6, so we return true.

My sol:

class Solution:

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

def t(root):

return root.val==root.right.val+root.left.val

return t(root)

**2239. Find Closest Number to Zero**

Given an integer array nums of size n, return the number with the value ***closest*** to 0 in nums. If there are multiple answers, return the number with the ***largest*** value.

**Example 1:**

**Input:** nums = [-4,-2,1,4,8]

**Output:** 1

**Explanation:**

The distance from -4 to 0 is |-4| = 4.

The distance from -2 to 0 is |-2| = 2.

The distance from 1 to 0 is |1| = 1.

The distance from 4 to 0 is |4| = 4.

The distance from 8 to 0 is |8| = 8.

Thus, the closest number to 0 in the array is 1.

My sol:

class Solution:

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

d=collections.Counter(nums)

a=-10000000

i=0

j=0

while a==-10000000:

if i in d:

a=i

break

if j in d:

a=j

break

i+=1

j-=1

return a

**2243. Calculate Digit Sum of a String**

You are given a string s consisting of digits and an integer k.

A **round** can be completed if the length of s is greater than k. In one round, do the following:

1. **Divide** s into **consecutive groups** of size k such that the first k characters are in the first group, the next k characters are in the second group, and so on. **Note** that the size of the last group can be smaller than k.
2. **Replace** each group of s with a string representing the sum of all its digits. For example, "346" is replaced with "13" because 3 + 4 + 6 = 13.
3. **Merge** consecutive groups together to form a new string. If the length of the string is greater than k, repeat from step 1.

Return s *after all rounds have been completed*.

**Example 1:**

**Input:** s = "11111222223", k = 3

**Output:** "135"

**Explanation:**

- For the first round, we divide s into groups of size 3: "111", "112", "222", and "23".

​​​​​Then we calculate the digit sum of each group: 1 + 1 + 1 = 3, 1 + 1 + 2 = 4, 2 + 2 + 2 = 6, and 2 + 3 = 5.

  So, s becomes "3" + "4" + "6" + "5" = "3465" after the first round.

- For the second round, we divide s into "346" and "5".

  Then we calculate the digit sum of each group: 3 + 4 + 6 = 13, 5 = 5.

  So, s becomes "13" + "5" = "135" after second round.

Now, s.length <= k, so we return "135" as the answer.

**My sol:**

class Solution:

def digitSum(self, s: str, k: int) -> str:

temp=[]

while len(s)>k:

for x in range(0,len(s),k):

temp.append(s[x:x+k])

s=0

t=''

for x in temp:

for y in x:

s+=int(y)

t+=str(s)

s=0

s=t

t=''

temp=[]

return s

**2248. Intersection of Multiple Arrays**

Given a 2D integer array nums where nums[i] is a non-empty array of **distinct** positive integers, return the list of integers that are present in ***each array*** of nums sorted in ***ascending order***.

**Example 1:**

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

**Output:** [3,4]

**Explanation:**

The only integers present in each of nums[0] = [**3**,1,2,**4**,5], nums[1] = [1,2,**3**,**4**], and nums[2] = [**3**,**4**,5,6] are 3 and 4, so we return [3,4].

**My sol:**

class Solution:

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

f=set(nums[0])

a=set(nums[0])

print(a)

for b in nums:

d=set(a).intersection(b)

f=set(d).intersection(f)

return sorted(list(f))

2251-2300 ( $$$$$ 11)

**2255. Count Prefixes of a Given String**

You are given a string array words and a string s, where words[i] and s comprise only of **lowercase English letters**.

Return *the****number of strings****in* words *that are a****prefix****of* s.

A **prefix** of a string is a substring that occurs at the beginning of the string. A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** words = ["a","b","c","ab","bc","abc"], s = "abc"

**Output:** 3

**Explanation:**

The strings in words which are a prefix of s = "abc" are:

"a", "ab", and "abc".

Thus the number of strings in words which are a prefix of s is 3.

class Solution:

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

co=0

for x in range(0,len(s)):

if s[:x+1] in words:

co+=words.count(s[:x+1])

return co

optimized:

count = 0

for word in words:

if s[:len(word)] == word:

count += 1

return count

**2259. Remove Digit From Number to Maximize Result**

You are given a string number representing a **positive integer** and a character digit.

Return *the resulting string after removing****exactly one occurrence****of*digit*from*number*such that the value of the resulting string in****decimal****form is****maximized***. The test cases are generated such that digit occurs at least once in number.

My sol:

class Solution:

def removeDigit(self, number: str, digit: str) -> str:

maxi=0

temp=[]

for x in range(0,len(number)):

if number[x]==digit:

maxi=max(maxi,int(number[:x]+number[x+1:]))

return str(maxi)

**2264. Largest 3-Same-Digit Number in String**

You are given a string num representing a large integer. An integer is **good** if it meets the following conditions:

* It is a **substring** of num with length 3.
* It consists of only one unique digit.

Return *the****maximum good****integer as a****string****or an empty string*""*if no such integer exists*.

**Example 1:**

**Input:** num = "6**777**133339"

**Output:** "777"

**Explanation:** There are two distinct good integers: "777" and "333".

"777" is the largest, so we return "777".

**My sol:**

class Solution:

def largestGoodInteger(self, num: str) -> str:

temp=[]

if '000' in num:

temp.append(000)

if '111' in num:

temp.append(111)

if '222' in num:

temp.append(222)

if '333' in num:

temp.append(333)

if '444' in num:

temp.append(444)

if '555' in num:

temp.append(555)

if '666' in num:

temp.append(666)

if '777' in num:

temp.append(777)

if '888' in num:

temp.append(888)

if '999' in num:

temp.append(999)

if len(temp)==0:

return ''

if str(max(temp))=='0':

return '000'

else:

return str(max(temp))

other:

class Solution:

def largestGoodInteger(self, num: str) -> str:

try:

lst = [str(x)\*3 for x in list(range(10))]

return max([i for i in lst if i in num ])

except:

return ''

**2269. Find the K-Beauty of a Number**

The **k-beauty** of an integer num is defined as the number of **substrings** of num when it is read as a string that meet the following conditions:

* It has a length of k.
* It is a divisor of num.

Given integers num and k, return *the k-beauty of*num.

Note:

* **Leading zeros** are allowed.
* 0 is not a divisor of any value.

A **substring** is a contiguous sequence of characters in a string.

**Example 1:**

**Input:** num = 240, k = 2

**Output:** 2

**Explanation:** The following are the substrings of num of length k:

- "24" from "**24**0": 24 is a divisor of 240.

- "40" from "2**40**": 40 is a divisor of 240.

Therefore, the k-beauty is 2.

class Solution:

def divisorSubstrings(self, num: int, k: int) -> int:

n=str(num)

count=0

for x in range(0,len(n)):

if int(n[x:x+k])!=0 and num%(int(n[x:x+k]))==0 and len(n[x:x+k])==k:

count+=1

return count

**2273. Find Resultant Array After Removing Anagrams**

You are given a **0-indexed** string array words, where words[i] consists of lowercase English letters.

In one operation, select any index i such that 0 < i < words.length and words[i - 1] and words[i] are **anagrams**, and **delete** words[i] from words. Keep performing this operation as long as you can select an index that satisfies the conditions.

Return words *after performing all operations*. It can be shown that selecting the indices for each operation in **any** arbitrary order will lead to the same result.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase using all the original letters exactly once. For example, "dacb" is an anagram of "abdc".

**Example 1:**

**Input:** words = ["abba","baba","bbaa","cd","cd"]

**Output:** ["abba","cd"]

**Explanation:**

One of the ways we can obtain the resultant array is by using the following operations:

- Since words[2] = "bbaa" and words[1] = "baba" are anagrams, we choose index 2 and delete words[2].

Now words = ["abba","baba","cd","cd"].

- Since words[1] = "baba" and words[0] = "abba" are anagrams, we choose index 1 and delete words[1].

Now words = ["abba","cd","cd"].

- Since words[2] = "cd" and words[1] = "cd" are anagrams, we choose index 2 and delete words[2].

Now words = ["abba","cd"].

We can no longer perform any operations, so ["abba","cd"] is the final answer.

My sol:

class Solution:

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

ad=[]

final=[]

ind=[0]

for x in words:

ad.append(collections.Counter(x))

# print(ad)

for x in range(1,len(ad)):

if ad[x]!=ad[x-1]:

final.append(ad[x])

ind.append(x)

print(ind)

f=[]

for x in ind:

f.append(words[x])

return f

**2278. Percentage of Letter in String**

Given a string s and a character letter, return the ***percentage*** of characters in s that equal letter ***rounded down*** to the nearest whole percent.

**Example 1:**

**Input:** s = "foobar", letter = "o"

**Output:** 33

**Explanation:**

The percentage of characters in s that equal the letter 'o' is 2 / 6 \* 100% = 33% when rounded down, so we return 33.

My sol :

def percentageLetter(self, s: str, letter: str) -> int:

den=0

num=0

for x in s:

den+=1

if x==letter:

num+=1

return int((num/den)\*100)

other sol :

return s.count(letter) \* 100 // len(s)

**2283. Check if Number Has Equal Digit Count and Digit Value**

**Example 1:**

**Input:** num = "1210"

**Output:** true

**Explanation:**

num[0] = '1'. The digit 0 occurs once in num.

num[1] = '2'. The digit 1 occurs twice in num.

num[2] = '1'. The digit 2 occurs once in num.

num[3] = '0'. The digit 3 occurs zero times in num.

The condition holds true for every index in "1210", so return true.

My sol :

def digitCount(self, num: str) -> bool:

i=0

final=''

for x in range(0,len(num)):

# print(num.count(str(x)))

if str(num.count(str(x)))!=num[i]:

return False

i+=1

return True

**2287. Rearrange Characters to Make Target String**

You are given two **0-indexed** strings s and target. You can take some letters from s and rearrange them to form new strings.

Return*the****maximum****number of copies of*target*that can be formed by taking letters from*s*and rearranging them.*

**Example 1:**

**Input:** s = "ilovecodingonleetcode", target = "code"

**Output:** 2

**Explanation:**

For the first copy of "code", take the letters at indices 4, 5, 6, and 7.

For the second copy of "code", take the letters at indices 17, 18, 19, and 20.

The strings that are formed are "ecod" and "code" which can both be rearranged into "code".

We can make at most two copies of "code", so we return 2.

My sol:

from collections import Counter

class Solution:

def rearrangeCharacters(self, s: str, target: str) -> int:

m=1000

for x in Counter(target).keys():

m=min((Counter(s)[x]//Counter(target)[x]),m)

return m

**2293. Min Max Game**

You are given a **0-indexed** integer array nums whose length is a power of 2.

Apply the following algorithm on nums:

1. Let n be the length of nums. If n == 1, **end** the process. Otherwise, **create** a new **0-indexed** integer array newNums of length n / 2.
2. For every **even** index i where 0 <= i < n / 2, **assign** the value of newNums[i] as min(nums[2 \* i], nums[2 \* i + 1]).
3. For every **odd** index i where 0 <= i < n / 2, **assign** the value of newNums[i] as max(nums[2 \* i], nums[2 \* i + 1]).
4. **Replace** the array nums with newNums.
5. **Repeat** the entire process starting from step 1.

Return *the last number that remains in*nums*after applying the algorithm.*

**Example 1:**

Diagram

Description automatically generated

**Input:** nums = [1,3,5,2,4,8,2,2]

**Output:** 1

**Explanation:** The following arrays are the results of applying the algorithm repeatedly.

First: nums = [1,5,4,2]

Second: nums = [1,4]

Third: nums = [1]

1 is the last remaining number, so we return 1.

My sol:

class Solution:

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

i=0

temp=[]

while len(nums)!=1:

for x in range(0,len(nums),2):

if i%2==0:

temp.append(min(nums[x],nums[x+1]))

else:

temp.append(max(nums[x],nums[x+1]))

# print(i,nums[x],nums[x+1])

i+=1

nums=temp

temp=[]

i=0

# print(temp)

# print(nums)

return nums[0]

**2295. Replace Elements in an Array**

You are given a **0-indexed** array nums that consists of n **distinct** positive integers. Apply m operations to this array, where in the ith operation you replace the number operations[i][0] with operations[i][1].

It is guaranteed that in the ith operation:

* operations[i][0] **exists** in nums.
* operations[i][1] does **not** exist in nums.

Return *the array obtained after applying all the operations*.

**Example 1:**

**Input:** nums = [1,2,4,6], operations = [[1,3],[4,7],[6,1]]

**Output:** [3,2,7,1]

**Explanation:** We perform the following operations on nums:

- Replace the number 1 with 3. nums becomes [**3**,2,4,6].

- Replace the number 4 with 7. nums becomes [3,2,**7**,6].

- Replace the number 6 with 1. nums becomes [3,2,7,**1**].

We return the final array [3,2,7,1].

My sol:

class Solution:

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

d={}

for i,j in enumerate(nums):

d[j]=i

for x in operations:

nums[d[x[0]]]=x[1]

d[x[1]]=d[x[0]]

return nums

**2299. Strong Password Checker II**

A password is said to be **strong** if it satisfies all the following criteria:

* It has at least 8 characters.
* It contains at least **one lowercase** letter.
* It contains at least **one uppercase** letter.
* It contains at least **one digit**.
* It contains at least **one special character**. The special characters are the characters in the following string: "!@#$%^&\*()-+".
* It does **not** contain 2 of the same character in adjacent positions (i.e., "aab" violates this condition, but "aba" does not).

Given a string password, return true*if it is a****strong****password*. Otherwise, return false.

**Example 1:**

**Input:** password = "IloveLe3tcode!"

**Output:** true

**Explanation:** The password meets all the requirements. Therefore, we return true.

My sol:

class Solution:

def strongPasswordCheckerII(self, password: str) -> bool:

if len(password)<8:

return False

flag=0

dummy=0

for x in password:

if x.lower()!=x:

flag=1

break

if flag==0:

return False

for x in password:

if x.upper()!=x:

dummy=1

break

if dummy==0:

return False

p=0

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

for x in arr:

if str(x) in password:

p=1

break

if p==0:

return False

mystr="!@#$%^&\*()-+"

s=0

for x in password:

if x in mystr:

s=1

break

if s==0:

return False

for x in range(0,len(password)-1):

if password[x]==password[x+1]:

return False

return True

2300-2344 ($$$$$ 9)

**2303. Calculate Amount Paid in Taxes**

You are given a **0-indexed** 2D integer array brackets where brackets[i] = [upperi, percenti] means that the ith tax bracket has an upper bound of upperi and is taxed at a rate of percenti. The brackets are **sorted** by upper bound (i.e. upperi-1 < upperi for 0 < i < brackets.length).

Tax is calculated as follows:

* The first upper0 dollars earned are taxed at a rate of percent0.
* The next upper1 - upper0 dollars earned are taxed at a rate of percent1.
* The next upper2 - upper1 dollars earned are taxed at a rate of percent2.
* And so on.

You are given an integer income representing the amount of money you earned. Return *the amount of money that you have to pay in taxes.* Answers within 10-5 of the actual answer will be accepted.

**Example 1:**

**Input:** brackets = [[3,50],[7,10],[12,25]], income = 10

**Output:** 2.65000

**Explanation:**

Based on your income, you have 3 dollars in the 1st tax bracket, 4 dollars in the 2nd tax bracket, and 3 dollars in the 3rd tax bracket.

The tax rate for the three tax brackets is 50%, 10%, and 25%, respectively.

In total, you pay $3 \* 50% + $4 \* 10% + $3 \* 25% = $2.65 in taxes.

My sol:

class Solution:

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

tax=0

p=0

d=[]

for x in brackets:

d.append(x[0]-p)

p=x[0]

# print(d)

temp=0

for x in range(0,len(d)):

# print(d[x],(brackets[x][1])/100)

if temp+d[x]<income:

tax+=d[x]\*((brackets[x][1])/100)

else:

tax+=(income-temp)\*((brackets[x][1])/100)

return tax

temp+=d[x]

return tax

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

Given a string of English letters s, return *the****greatest****English letter which occurs as****both****a lowercase and uppercase letter in* s. The returned letter should be in **uppercase**. If no such letter exists, return *an empty string*.

An English letter b is **greater** than another letter a if b appears **after** a in the English alphabet.

**Example 1:**

**Input:** s = "l**Ee**TcOd**E**"

**Output:** "E"

**Explanation:**

The letter 'E' is the only letter to appear in both lower and upper case.

My sol:

class Solution:

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

arr=['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']

for var in range(len(arr)-1,-1,-1):

if str(arr[var]) in s and str(arr[var]).upper() in s:

return arr[var].upper()

return ""

**2315. Count Asterisks**

You are given a string s, where every **two** consecutive vertical bars '|' are grouped into a **pair**. In other words, the 1st and 2nd '|' make a pair, the 3rd and 4th '|' make a pair, and so forth.

Return *the number of*'\*'*in*s*,****excluding****the*'\*'*between each pair of*'|'.

**Note** that each '|' will belong to **exactly** one pair.

**Example 1:**

**Input:** s = "l|\*e\*et|c\*\*o|\*de|"

**Output:** 2

**Explanation:** The considered characters are underlined: "l|\*e\*et|c\*\*o|\*de|".

The characters between the first and second '|' are excluded from the answer.

Also, the characters between the third and fourth '|' are excluded from the answer.

There are 2 asterisks considered. Therefore, we return 2.

My sol:

class Solution:

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

a=s.split('|')

print(a)

ans=0

for x in range(0,len(a)):

if x %2==0:

print(a[x])

ans+=a[x].count('\*')

return ans

**2319. Check if Matrix Is X-Matrix**

A square matrix is said to be an **X-Matrix** if **both** of the following conditions hold:

1. All the elements in the diagonals of the matrix are **non-zero**.
2. All other elements are 0.

Given a 2D integer array grid of size n x n representing a square matrix, return true*if*grid*is an X-Matrix*. Otherwise, return false.

**Example 1:**

Calendar

Description automatically generated

**Input:** grid = [[2,0,0,1],[0,3,1,0],[0,5,2,0],[4,0,0,2]]

**Output:** true

**Explanation:** Refer to the diagram above.

An X-Matrix should have the green elements (diagonals) be non-zero and the red elements be 0.

Thus, grid is an X-Matrix.

My sol:

class Solution:

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

for x in range(0,len(grid)):

for y in range(0,len(grid[x])):

if x==y and grid[x][y]==0:

return False

for x in range(0,len(grid)):

for y in range(0,len(grid[x])):

if x+y==len(grid)-1 and grid[x][y]==0:

return False

for x in range(0,len(grid)):

for y in range(0,len(grid[x])):

if x!=y and grid[x][y]!=0 and x+y!=len(grid)-1:

return False

return True

**2323. Find Minimum Time to Finish All Jobs II**

You are given two **0-indexed** integer arrays jobs and workers of **equal** length, where jobs[i] is the amount of time needed to complete the ith job, and workers[j] is the amount of time the jth worker can work each day.

Each job should be assigned to **exactly** one worker, such that each worker completes **exactly** one job.

Return *the****minimum****number of days needed to complete all the jobs after assignment.*

**Example 1:**

**Input:** jobs = [5,2,4], workers = [1,7,5]

**Output:** 2

**Explanation:**

- Assign the 2nd worker to the 0th job. It takes them 1 day to finish the job.

- Assign the 0th worker to the 1st job. It takes them 2 days to finish the job.

- Assign the 1st worker to the 2nd job. It takes them 1 day to finish the job.

It takes 2 days for all the jobs to be completed, so return 2.

It can be proven that 2 days is the minimum number of days needed.

My sol:

class Solution:

def minimumTime(self, jobs: List[int], workers: List[int]) -> int:

jobs.sort()

workers.sort()

a=0

for x in range(0,len(jobs)):

a=max(a,ceil(jobs[x]/workers[x]))

return a

**2331. Evaluate Boolean Binary Tree (dfs)**

You are given the root of a **full binary tree** with the following properties:

* **Leaf nodes** have either the value 0 or 1, where 0 represents False and 1 represents True.
* **Non-leaf nodes** have either the value 2 or 3, where 2 represents the boolean OR and 3 represents the boolean AND.

The **evaluation** of a node is as follows:

* If the node is a leaf node, the evaluation is the **value** of the node, i.e. True or False.
* Otherwise, **evaluate** the node's two children and **apply** the boolean operation of its value with the children's evaluations.

Return*the boolean result of****evaluating****the*root*node.*

A **full binary tree** is a binary tree where each node has either 0 or 2 children.

A **leaf node** is a node that has zero children.

**Example 1:**

Diagram

Description automatically generated

**Input:** root = [2,1,3,null,null,0,1]

**Output:** true

**Explanation:** The above diagram illustrates the evaluation process.

The AND node evaluates to False AND True = False.

The OR node evaluates to True OR False = True.

The root node evaluates to True, so we return true.

**My sol:**

class Solution:

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

def evaluate(root):

if root:

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

return root.val

left = evaluate(root.left)

right = evaluate(root.right)

if root.val == 2:

return (left or right)

if root.val == 3:

return (left and right)

return evaluate(root)

**2325. Decode the Message**

You are given the strings key and message, which represent a cipher key and a secret message, respectively. The steps to decode message are as follows:

1. Use the **first** appearance of all 26 lowercase English letters in key as the **order** of the substitution table.
2. Align the substitution table with the regular English alphabet.
3. Each letter in message is then **substituted** using the table.
4. Spaces ' ' are transformed to themselves.

* For example, given key = "**hap**p**y** **bo**y" (actual key would have **at least one** instance of each letter in the alphabet), we have the partial substitution table of ('h' -> 'a', 'a' -> 'b', 'p' -> 'c', 'y' -> 'd', 'b' -> 'e', 'o' -> 'f').

Return *the decoded message*.

**Example 1:**

A picture containing shape

Description automatically generated

**Input:** key = "the quick brown fox jumps over the lazy dog", message = "vkbs bs t suepuv"

**Output:** "this is a secret"

**Explanation:** The diagram above shows the substitution table.

It is obtained by taking the first appearance of each letter in "**the** **quick** **brown** **f**o**x** **j**u**mps** o**v**er the **lazy** **d**o**g**".

class Solution:

def decodeMessage(self, key: str, message: str) -> str:

der=['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']

temp=key

res=""

d={}

dic=""

temp=temp.replace(' ','')

for x in temp:

if x not in dic:

dic=dic+x

else:

pass

print(dic)

ans=""

for x in range(0,len(message)):

if message[x]!=" ":

ans=ans+(der[dic.index(message[x])])

else:

ans=ans+" "

return ans

**2335. Minimum Amount of Time to Fill Cups**

You have a water dispenser that can dispense cold, warm, and hot water. Every second, you can either fill up 2 cups with **different** types of water, or 1 cup of any type of water.

You are given a **0-indexed** integer array amount of length 3 where amount[0], amount[1], and amount[2] denote the number of cold, warm, and hot water cups you need to fill respectively. Return *the****minimum****number of seconds needed to fill up all the cups*.

**Example 1:**

**Input:** amount = [1,4,2]

**Output:** 4

**Explanation:** One way to fill up the cups is:

Second 1: Fill up a cold cup and a warm cup.

Second 2: Fill up a warm cup and a hot cup.

Second 3: Fill up a warm cup and a hot cup.

Second 4: Fill up a warm cup.

It can be proven that 4 is the minimum number of seconds needed.

My sol:

class Solution:

def fillCups(self, amount: List[int]) -> int:

return max(max(amount), (sum(amount) + 1) // 2)

**2341. Maximum Number of Pairs in Array**

You are given a **0-indexed** integer array nums. In one operation, you may do the following:

* Choose **two** integers in nums that are **equal**.
* Remove both integers from nums, forming a **pair**.

The operation is done on nums as many times as possible.

Return *a****0-indexed****integer array*answer*of size*2*where*answer[0]*is the number of pairs that are formed and*answer[1]*is the number of leftover integers in*nums*after doing the operation as many times as possible*.

**Example 1:**

**Input:** nums = [1,3,2,1,3,2,2]

**Output:** [3,1]

**Explanation:**

Form a pair with nums[0] and nums[3] and remove them from nums. Now, nums = [3,2,3,2,2].

Form a pair with nums[0] and nums[2] and remove them from nums. Now, nums = [2,2,2].

Form a pair with nums[0] and nums[1] and remove them from nums. Now, nums = [2].

No more pairs can be formed. A total of 3 pairs have been formed, and there is 1 number leftover in nums.

My sol:

class Solution:

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

count=0

sum1=0

d= collections.Counter(nums)

for k,v in d.items():

sum1+=v//2

if v%2==0:

pass

else:

count+=1

return [sum1,count]

**2347. Best Poker Hand**

You are given an integer array ranks and a character array suits. You have 5 cards where the ith card has a rank of ranks[i] and a suit of suits[i].

The following are the types of **poker hands** you can make from best to worst:

1. "Flush": Five cards of the same suit.
2. "Three of a Kind": Three cards of the same rank.
3. "Pair": Two cards of the same rank.
4. "High Card": Any single card.

Return *a string representing the****best****type of****poker hand****you can make with the given cards.*

**Note** that the return values are **case-sensitive**.

**Example 1:**

**Input:** ranks = [13,2,3,1,9], suits = ["a","a","a","a","a"]

**Output:** "Flush"

**Explanation:** The hand with all the cards consists of 5 cards with the same suit, so we have a "Flush".

**My sol:**

class Solution:

def bestHand(self, ranks: List[int], suits: List[str]) -> str:

if len(list(set(suits)))==1:

return "Flush"

d=collections.Counter(ranks)

for k,v in d.items():

if v >= 3:

return "Three of a Kind"

for k,v in d.items():

if v >= 2:

return "Pair"

return "High Card"

**2351. First Letter to Appear Twice**

Given a string s consisting of lowercase English letters, return *the first letter to appear****twice***.

**Note**:

* A letter a appears twice before another letter b if the **second** occurrence of a is before the **second** occurrence of b.
* s will contain at least one letter that appears twice.

**Example 1:**

**Input:** s = "abccbaacz"

**Output:** "c"

**Explanation:**

The letter 'a' appears on the indexes 0, 5 and 6.

The letter 'b' appears on the indexes 1 and 4.

The letter 'c' appears on the indexes 2, 3 and 7.

The letter 'z' appears on the index 8.

The letter 'c' is the first letter to appear twice, because out of all the letters the index of its second occurrence is the smallest.

**My sol:**

class Solution:

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

d={}

for x in s:

if x in d:

d[x]+=1

else:

d[x] = 1

if d[x]==2:

return x