1. **202-Happy Number**

class Solution(object):

def isHappy(self, n):

"""

:type n: int

:rtype: bool

"""

set1=set()

while n not in set1:

set1.add(n)

temp=0

while n:

n,val=divmod(n,10)

temp+=val\*\*2

n=temp

return True if n==1 else False

1. **204-CountPrimes**

class Solution:

def countPrimes(self, n):

"""

:type n: int

:rtype: int

"""

if n<3:

return 0

primes=[1]\*n

primes[0]=primes[1]=0

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

if primes[i]:

primes[i\*i:n:i]=[0]\*((n-1)//i-i+1)

return sum(primes)

1. **205-Isomorphic Strings**

class Solution:

def isIsomorphic(self, s, t):

"""

:type s: str

:type t: str

:rtype: bool

"""

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

return False

dic={}

set1=set()

for i in range(len(s)):

if s[i] in dic:

if t[i]!=dic[s[i]]:

return False

elif t[i] in set1:

return False

else:

dic[s[i]]=t[i]

set1.add(t[i])

return True

1. **217-ContainsDuplicate** class Solution:

def containsDuplicate(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

return len(nums)!=len(set(nums))

1. **219-Contains Duplicate II**

class Solution:

def containsNearbyDuplicate(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: bool

"""

dic={}

for i,n in enumerate(nums):

if n in dic and i-dic[n]<=k:

return True

dic[n]=i

return False

1. **231-Power of Two** class Solution(object):

def isPowerOfTwo(self, n):

"""

:type n: int

:rtype: bool

"""

if n==0:

return False

while n%2==0:

n/=2

return n==1

**7. 242-Valid Anagram**

class Solution:

def isAnagram(self, s, t):

"""

:type s: str

:type t: str

:rtype: bool

"""

cnt1=collections.Counter(s)

cnt2=collections.Counter(t)

return not (cnt1-cnt2) and not (cnt2-cnt1)

**8. 258-Add Digits**

class Solution:

def addDigits(self, num):

"""

:type num: int

:rtype: int

"""

return (num-1)%9+1 if num else 0

**9. 263-Ugly Number**

class Solution(object):

def isUgly(self, num):

"""

:type num: int

:rtype: bool

"""

if num<1:

return False

for i in [2,3,5]:

while num%i==0: num/=i

return num==1

**10. 268-Missing Number**

class Solution(object):

def missingNumber(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

l=len(nums)

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

**11. 283-Move Zeroes**

class Solution:

def moveZeroes(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

cnt=0

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

if nums[i]==0:

del nums[i] cnt+=1

nums+=[0]\*cnt

**12. 290-Word Pattern**

class Solution:

def wordPattern(self, pattern, str):

"""

:type pattern: str

:type str: str

:rtype: bool

"""

frac=str.split()

return len(set(zip(pattern,frac)))==len(set(pattern)) and len(set(zip(pattern,frac))) ==len(set(frac)) and len(frac)==len(pattern)

**13. 292-Nim Game**

class Solution(object):

def canWinNim(self, n):

"""

:type n: int

:rtype: bool

"""

return True if n%4 else False

**14. 326-Power of Three**

class Solution(object):

def isPowerOfThree(self, n):

"""

:type n: int

:rtype: bool

"""

return n>0 and 1162261467%n==0

**15. 342-Power of Four**

class Solution(object):

def isPowerOfFour(self, num):

"""

:type num: int

:rtype: bool

"""

tmp=1

while tmp<num:

tmp\*=4

return num==tmp

**16. 344-Reverse String**

class Solution(object):

def reverseString(self, s):

"""

:type s: str

:rtype: str

"""

return s[::-1]

**17. 345-Reverse Vowels of a String**

class Solution:

def reverseVowels(self, s):

"""

:type s: str

:rtype: str

"""

s=list(s)

vowels=set("aeiouAEIOU")

l=0

r=len(s)-1

while l<r:

while s[l] not in vowels and l<r:

l+=1

while s[r] not in vowels and l<r:

r-=1

s[l],s[r]=s[r],s[l]

l+=1

r-=1

return ''.join(s)

**18. 349-Intersection of Two Arrays**

class Solution:

def intersection(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: List[int]

"""

return list(set(nums1)&set(nums2))

**19. 350-Intersection of Two Arrays II**

class Solution:

def intersect(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: List[int]

"""

dic=collections.Counter(nums1)&collections.Counter(nums2)

return [x for i in dic for x in [i]\*dic[i]]

**20. 367-Valid Perfect Square**

class Solution(object):

def isPerfectSquare(self, num):

"""

:type num: int

:rtype: bool

"""

l,r=1,num

while l<r:

m=(l+r)/2

if m\*\*2<num:

l=m+1

else:

r=m

return l\*\*2==num

**21. 383-Ransom Note**

class Solution:

def canConstruct(self, ransomNote, magazine):

"""

:type ransomNote: str

:type magazine: str

:rtype: bool

"""

dic1=collections.Counter(ransomNote)

dic2=collections.Counter(magazine)

return not dic1-dic2

**22. 387-First Unique Character in a String**

class Solution:

def firstUniqChar(self, s):

"""

:type s: str

:rtype: int

"""

dic=collections.Counter(s)

for i,c in enumerate(s):

if dic[c]==1:

return i

return -1

**23. 389-Find the Difference**

class Solution(object):

def findTheDifference(self, s, t):

"""

:type s: str

:type t: str

:rtype: str

"""

return list(collections.Counter(t)-collections.Counter(s))[0]

**24. 400-Nth Digit**

class Solution(object):

def findNthDigit(self, n):

"""

:type n: int

:rtype: int

"""

if n<10:

return n

i,p=1,9

while True:

n+=p

p=p\*10+9

i+=1

if n<p\*i:

return int(str((n+i-1)//i)[(n+i-1)%i])

**25. 409-Longest Palindrome**

class Solution:

def longestPalindrome(self, s):

"""

:type s: str

:rtype: int

"""

dic=collections.Counter(s)

ans=0

mark=0

for c in dic:

if dic[c]%2==0:

ans+=dic[c]

else:

mark=1 ans+=(dic[c]-1)

return ans+mark

**26. 414-Third Maximum Number**

class Solution:

def thirdMax(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

ans=[float('-inf'),float('-inf'),float('-inf')]

for num in nums:

if num not in ans:

if num>ans[0]:

ans=[num,ans[0],ans[1]]

elif num>ans[1]:

ans[1:]=[num,ans[1]]

elif num>ans[2]:

ans[2]=num

return ans[2] if ans[2]!=float('-inf') else ans[0]

**27. 434-Number of Segments in a String**

class Solution:

def countSegments(self, s):

"""

:type s: str

:rtype: int

"""

return len(s.split())

**28. 438-Find All Anagrams in a String**

class Solution:

def findAnagrams(self, s, p):

"""

:type s: str

:type p: str

:rtype: List[int]

"""

cnt1=[0]\*26

cnt2=[0]\*26

l=len(p)

for c in p:

cnt1[ord(c)-97]+=1 for c in s[:l-1]:

cnt2[ord(c)-97]+=1 ans=[]

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

cnt2[ord(s[i+l-1])-97]+=1

if cnt1==cnt2:

ans.append(i)

cnt2[ord(s[i])-97]-=1

return ans

**29. 441-Arranging Coins**

class Solution(object):

def arrangeCoins(self, n):

"""

:type n: int

:rtype: int

"""

return int(math.sqrt(n\*2+0.25)-0.5)

**30. 443-String Compression**

class Solution:

def compress(self, chars):

"""

:type chars: List[str]

:rtype: int

"""

cur=chars[0]

cnt=0

ans=[]

for c in chars+[' ']:

if c==cur:

cnt+=1

else:

ans+=[cur]

if cnt!=1:

ans+=list(str(cnt))

cnt=1

cur=c

chars[:]=ans

return len(chars)

**31. 448-Find All Numbers Disappeared in an Array**

class Solution:

def findDisappearedNumbers(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

return list(set(range(1,len(nums)+1))-set(nums))

**32. 453-Minimum Moves to Equal Array Elements**

class Solution(object):

def minMoves(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

return sum(nums)-min(nums)\*len(nums)

**33. 455-Assign Cookies**

class Solution:

def findContentChildren(self, g, s):

"""

:type g: List[int]

:type s: List[int]

:rtype: int

"""

g.sort()

s.sort()

l=len(g)

ans=0

for x in s:

if ans<l and x>=g[ans]:

ans+=1

return ans

**34. 459-Repeated Substring Pattern**

class Solution:

def repeatedSubstringPattern(self, s):

"""

:type s: str

:rtype: bool

"""

return s in (s+s)[1:-1]

**35. 461-Hamming Distance**

class Solution:

def hammingDistance(self, x, y):

"""

:type x: int

:type y: int

:rtype: int

"""

return bin(x^y).count('1')

**36. 475-Heaters**

class Solution:

def findRadius(self, houses, heaters):

"""

:type houses: List[int]

:type heaters: List[int]

:rtype: int

"""

ans=0

heaters=[-math.inf]+sorted(heaters)+[math.inf]

houses.sort()

i=0

for house in houses:

while house>heaters[i]:

i+=1

ans=max(ans,min(heaters[i]-house,house-heaters[i-1]))

return ans

**37. 476-Number Complement**

class Solution:

def findComplement(self, num):

"""

:type num: int

:rtype: int

"""

return pow(2,num.bit\_length())-num-1

**38. 479-Largest Palindrome Product**

class Solution:

def largestPalindrome(self, n):

"""

:type n: int

:rtype: int

"""

if n==1:

return 9

for a in range(2,10\*\*n):

left=10\*\*n-a

right=int(str(left)[::-1])

if a\*\*2-4\*right>=0:

x=a-(a\*\*2-4\*right)\*\*0.5

if x//2==x/2:

return (10\*\*n\*left+right)%1337

**39. 482-License Key Formatting**

class Solution:

def licenseKeyFormatting(self, S, K):

"""

:type S: str

:type K: int

:rtype: str

"""

S=S.replace('-','').upper()[::-1]

ans=[]

for i in range(0,len(S),K):

ans.append(S[i:i+K])

return '-'.join(ans)[::-1]

**40. 485-Max Consecutive Ones**

class Solution:

def findMaxConsecutiveOnes(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

ans=0

tmp=0

nums.append(0)

for num in nums:

if num:

tmp+=1

else:

ans=max(ans,tmp)

tmp=0

return ans

**41. 496-Next Greater Element I**

class Solution:

def nextGreaterElement(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: List[int]

"""

dic={}

stack=[]

for i in nums2:

while stack and stack[-1] < i:

dic[stack.pop()] = i

stack.append(i)

return [dic.get(i, -1) for i in nums1]

**42. 507-Perfect Number**

class Solution(object):

def checkPerfectNumber(self, num):

"""

:type num: int

:rtype: bool

"""

return num in [6,28,496,8128,33550336]

**43. 628-Maximum Product of Three Numbers**

class Solution(object):

def maximumProduct(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

l,s=heapq.nlargest(3,nums),heapq.nsmallest(2,nums)

x,y=l[1]\*l[2],s[0]\*s[1]

ans=max(x,y)\*l[0]

return ans

**44. 633-Sum of Square Numbers**

class Solution(object):

def judgeSquareSum(self, c):

"""

:type c: int

:rtype: bool

"""

l=0

r=int(c\*\*0.5)

while l<=r:

t=l\*l+r\*r

if t==c:

return True

elif t<c:

l+=1

else:

r-=1

return False

**45. 672-Bulb Switcher II**

class Solution:

def flipLights(self, n, m):

"""

:type n: int

:type m: int

:rtype: int

"""

if m==0 or n==0:

return 1

if n==1:

return 2

if n==2:

if m==1:

return 3

return 4

if n>=3:

if m==1:

return 4

if m==2:

return 7

return 8

**46. 728-Self Dividing Numbers**

class Solution(object):

def selfDividingNumbers(self, left, right):

"""

:type left: int

:type right: int

:rtype: List[int]

"""

ans=[]

for x in range(left,right+1): y=x

while y: val=y%10

if val==0 or x%(val):

break

y//=10

if not y:

ans.append(x)

return ans

**47. 754-Reach a Number**

class Solution:

def reachNumber(self, target):

"""

:type target: int

:rtype: int

"""

target=abs(target)

x=math.ceil((2\*target+0.25)\*\*0.5-0.5)

return x if (x\*(x+1)/2-target)%2==0 else x+1+x%2

**48. 877-Stone Game**

class Solution:

def stoneGame(self, piles):

"""

:type piles: List[int]

:rtype: bool

"""

return True

**49. 914-X of a Kind in a Deck of Cards**

class Solution:

def hasGroupsSizeX(self, deck):

"""

:type deck: List[int]

:rtype: bool

"""

dic=collections.defaultdict(int)

for i in deck:

dic[i]+=1

x=dic[deck[0]]

for i in dic.values():

while i%x!=0:

i,x=x,i%x

return x!=1

**50. 942-DI String Match**

class Solution:

def diStringMatch(self, S):

"""

:type S: str

:rtype: List[int]

"""

l=0

r=len(S)

ans=[]

for c in S:

if c=='I':

ans.append(l)

l+=1

else:

ans.append(r)

r-=1

ans.append(l)

return ans

**51. 7- Reverse Integer**

class Solution(object):

def reverse(self, x):

"""

:type x: int

:rtype: int

"""

if x < 0:

x = str(x)[1:]

y = -int(str(x)[::-1])

else:

y = int(str(x)[::-1])

if y < -2 \*\* 31 or y > 2 \*\* 31 - 1:

return 0

else:

return y

**52. 9-Palindrome Number** class Solution:

def isPalindrome(self, x):

if str(x) == str(x)[::-1]:

return True

else:

return False

**53. 11-Container With Most Water** class Solution:

def maxArea(self, height):

begin = 0

end = len(height)-1

ans = 0

while begin < end:

ans = max(ans,(end-begin) \* min(height[begin],height[end]))

if height[begin] > height[end]:

end -= 1

else:

begin += 1

return ans

**54.5512-Integer to Roman**

class Solution:

def intToRoman(self, num):

unit = ['','I','II','III','IV','V','VI','VII','VIII','IX']

ten = ['','X','XX','XXX','XL','L','LX','LXX','LXXX','XC']

hundred = ['','C','CC','CCC','CD','D','DC','DCC','DCCC','CM']

thousand = ['','M','MM','MMM']

d = [unit,ten,hundred,thousand]

num = str(num)[::-1]

ans = []

for i in range(len(num)):

ans.append(d[i][int(num[i])])

return "".join(ans[::-1])

**55. 14-Longest Common Prefix**

class Solution:

def longestCommonPrefix(self, strs):

if len(strs) == 0:

return ''

for i in range(len(strs[0])):

letter = strs[0][i]

for string in strs:

if len(string) <= i or string[i] != letter:

return strs[0][:i]

return strs[0]

**56. 26-Remove Duplicates from Sorted Array**

class Solution:

def removeDuplicates(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

if len(nums) == 0:

return 0

cnt = 1

mark = nums[0]

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

if nums[i] != mark:

nums[cnt] = nums[i]

mark = nums[i]

cnt += 1

return cnt

**57. 27-Remove Element**

class Solution:

def removeElement(self, nums, val):

"""

:type nums: List[int]

:type val: int

:rtype: int

"""

cnt = 0

for i in range(len(nums)):

if nums[i] != val:

nums[cnt] = nums[i]

cnt += 1

return cnt

**58. 28-Implement strStr()** class Solution:

def strStr(self, haystack, needle):

if needle:

return haystack.find(needle)

else:

return 0

**59. 34-Find First and Last Position of Element in Sorted Array** class Solution:

"""

* + returns leftmost (or rightmost) index at which `target` should be inserted in sorted

array `nums` via binary search.

"""

def extreme\_insertion\_index(self, nums, target, left):

lo = 0

hi = len(nums)

while lo < hi:

mid = (lo + hi) // 2

if nums[mid] > target or (left and target == nums[mid]):

hi = mid

else:

lo = mid+1

return lo

def searchRange(self, nums, target):

left\_idx = self.extreme\_insertion\_index(nums, target, True)

"""

1. assert that `left\_idx` is within the array bounds and that `target`
2. is actually in `nums`.

"""

if left\_idx == len(nums) or nums[left\_idx] != target:

return [-1, -1]

return [left\_idx, self.extreme\_insertion\_index(nums, target, False)-1]

**60. 35-Search Insert Position** class Solution:

def searchInsert(self, nums, target):

if nums[-1] < target:

return len(nums)

for i in range(len(nums)):

if nums[i] == target or nums[i] > target:

return i

**61. 50-Pow(x, n)**

class Solution:

def myPow(self, x, n):

"""

:type x: float

:type n: int

:rtype: float

"""

if n < 0:

x = 1./x

n = -n

power = 1

while n:

if n & 1:

power \*= x

x \*= x

n >>= 1

return power

**62. 58-Length of Last Word** class Solution:

def lengthOfLastWord(self, s): s = s.rstrip()

if not s:

return 0

index = len(s) - 1

cnt = 0

while index >= 0 and s[index] != ' ':

cnt += 1

index -= 1

return cnt

**63. 63-Unique Paths II**

class Solution:

def uniquePathsWithObstacles(self, obstacleGrid):

"""

:type obstacleGrid: List[List[int]]

:rtype: int

"""

if obstacleGrid[0][0] == 1:

return 0

width,length = len(obstacleGrid), len(obstacleGrid[0])

dp = [0 for j in range(length+1)]

dp[1] = 1

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

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

if i==1 and j==1:

continue

if obstacleGrid[i-1][j-1] == 0:

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

else:

dp[j] = 0

return dp[length]

**64. 64-Minimum Path Sum**

class Solution:

def minPathSum(self, grid):

"""

:type grid: List[List[int]]

:rtype: int

"""

* + #get the rows number
  + m = len(grid)
  + #get the columns number
  + n = len(grid[0])
  + #calculate the 1st column
  + for i in range(1,m):

grid[i][0] = grid[i][0] + grid[i-1][0]

* + #calculate the 1st row

for i in range(1,n):

grid[0][i] = grid[0][i] + grid[0][i-1]

for i in range(1, m):

f or j in range(1, n):

grid[i][j] += grid[i-1][j] if grid[i-1][j]<grid[i][j-1] else grid[i][j-1]

return grid[m-1][n-1]

**65. 66-Plus One**

class Solution(object):

def plusOne(self, digits):

"""

:type digits: List[int]

:rtype: List[int]

"""

pos = len(digits) - 1

while True:

temp = digits[pos] + 1

if temp > 9:

digits[pos] = temp % 10

pos -= 1

if pos == -1:

digits.insert(0,1)

return digits

else:

digits[pos] = temp

return digits

**66. 67-Add Binary**

class Solution(object):

def addBinary(self, a, b):

"""

:type a: str

:type b: str

:rtype: str

"""

return bin(int(a,2) + int(b,2))[2:]

**67. 69-Sqrt(x)**

class Solution(object):

def mySqrt(self, x):

"""

:type x: int

:rtype: int

"""

return int(math.sqrt(x))

**68. 70-Climbing Stairs** class Solution:

def climbStairs(self, n):

"""

:type n: int

:rtype: int

"""

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

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

res[i] = res[i-1] + res[i-2]

return res[n]

**69. 88-Merge Sorted Array**

class Solution:

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

end\_pos = m + n - 1

while m > 0 and n > 0:

if nums1[m-1] > nums2[n-1]:

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

m -= 1

else:

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

n -= 1

if n > 0:

nums1[:n] = nums2[:n]

**70. 121-Best Time to Buy and Sell Stock**

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

if prices == []: return 0

ans = 0

min\_num = prices[0]

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

min\_num = min(prices[i],min\_num)

ans = max(ans,prices[i]-min\_num)

return ans

**71. 125-Valid Palindrome**

class Solution(object):

def isPalindrome(self, s):

"""

:type s: str

:rtype: bool

"""

temp = []

for i in s:

if i.isalnum():

temp.append(i.lower())

return temp == temp[::-1]

**72. 136-Single Number**

class Solution(object):

def singleNumber(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

res = 0

for i in nums:

res ^= i

return res

class Solution(object):

def singleNumber(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

return reduce(lambda x,y:x^y , nums)

**73. 137-Single Number II**

class Solution(object):

def singleNumber(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

d = dict()

for i in nums:

if i not in d:

d[i] = 1

else:

d[i] += 1

for item in d:

if d[item] == 1:

return item

class Solution(object):

def singleNumber(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

return (sum(set(nums)) \* 3 - sum(nums)) // 2

**74. 167-Two Sum II - Input array is sorted**

class Solution(object):

def twoSum(self, numbers, target):

"""

:type numbers: List[int]

:type target: int

:rtype: List[int]

"""

d = dict(zip(numbers,range(1,len(numbers)+1)))

for i in numbers:

if target-i in d:

if target-i == i:

return [d[i]-1,d[i]]

return [d[i],d[target-i]]

**75. 169-Majority Element**

class Solution(object):

def majorityElement(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

maj = nums[0] cnt = 1

for i in nums[1:]: if i == maj:

cnt += 1

else:

if cnt == 0:

maj = i

cnt = 1

else:

cnt -= 1

return maj

**76. 172-Factorial Trailing Zeroes**

class Solution(object):

def trailingZeroes(self, n):

"""

:type n: int

:rtype: int

"""

cnt = 0

while n >= 5:

n -= (n % 5)

n /= 5

cnt += n

return cnt

**77. 189-Rotate Array**

class Solution(object):

def reverseOrder(self,nums,begin,end):

while begin < end:

nums[begin],nums[end] = nums[end],nums[begin] begin += 1

end -=1

def rotate(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: void Do not return anything, modify nums in-place instead.

"""

k %= len(nums)

self.reverseOrder(nums,0,len(nums)-k-1) self.reverseOrder(nums,len(nums)-k,len(nums)-1) self.reverseOrder(nums,0,len(nums)-1)

**78. 190-Reverse Bits**

class Solution:

# @param n, an integer

# @return an integer

def reverseBits(self, n):

temp = ("0" \* 32 + bin(n)[2:])[-32:][::-1]

return int(temp,2)

class Solution:

* + @param n, an integer
  + @return an integer

def reverseBits(self, n):

temp = bin(n)[2:].zfill(32)[::-1]

return int(temp,2)

1. **191-Number of 1 Bits**

class Solution(object):

def hammingWeight(self, n):

"""

:type n: int

:rtype: int

"""

return bin(n).count('1')

**80. 746-Min Cost Climbing Stairs**

class Solution(object):

def minCostClimbingStairs(self, cost):

"""

:type cost: List[int]

:rtype: int

"""

if len(cost) == 0 or len(cost) == 1:

return 1

a, b = cost[0], cost[1]

for index in range(2, len(cost)):

a, b = b, cost[index] + min(a, b)

return min(a,b)

**81. 867-Transpose Matrix**

class Solution(object):

def transpose(self, A):

R, C = len(A), len(A[0])

ans = [[None] \* R for \_ in xrange(C)]

for r, row in enumerate(A):

for c, val in enumerate(row):

ans[c][r] = val

return ans

**82. 868-Binary Gap**

class Solution(object):

def binaryGap(self, N):

A = [i for i in xrange(32) if (N >> i) & 1]

if len(A) < 2:

return 0

return max(A[i+1] - A[i] for i in xrange(len(A) - 1))

**83. 884-Uncommon Words from Two Sentences**

class Solution(object):

def uncommonFromSentences(self, A, B):

count = {}

for word in A.split():

count[word] = count.get(word, 0) + 1

for word in B.split():

count[word] = count.get(word, 0) + 1

#Alternatively:

#count = collections.Counter(A.split())

#count += collections.Counter(B.split())

return [word for word in count if count[word] == 1]

**84. 888-Fair Candy Swap**

class Solution(object):

def fairCandySwap(self, A, B): Sa, Sb = sum(A), sum(B) setB = set(B)

for x in A:

if x + (Sb - Sa) / 2 in setB:

return [x, x + (Sb - Sa) / 2]

**85. 893-Groups of Special-Equivalent Strings**

class Solution(object):

def numSpecialEquivGroups(self, A):

def count(A):

ans = [0] \* 52

for i, letter in enumerate(A):

ans[ord(letter) - ord('a') + 26 \* (i%2)] += 1

return tuple(ans)

return len({count(word) for word in A})

**86. 896-Monotonic Array**

class Solution(object):

def isMonotonic(self, A):

return (all(A[i] <= A[i+1] for i in xrange(len(A) - 1)) or

all(A[i] >= A[i+1] for i in xrange(len(A) - 1)))

**87. 905-Sort Array By Parity**

class Solution(object):

def sortArrayByParity(self, A):

A.sort(key = lambda x: x % 2)

return A

**88. 908-Smallest Range I**

class Solution(object):

def smallestRangeI(self, A, K):

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

**89. 914-X of a Kind in a Deck of Cards**

class Solution(object):

def hasGroupsSizeX(self, deck):

from fractions import gcd

vals = collections.Counter(deck).values()

return reduce(gcd, vals) >= 2

**90. 917-Reverse Only Letters**

class Solution(object):

def reverseOnlyLetters(self, S): letters = [c for c in S if c.isalpha()] ans = []

for c in S:

if c.isalpha():

ans.append(letters.pop())

else:

ans.append(c)

return “".join(ans)

**91. 921-Minimum Add to Make Parentheses Valid**

class Solution(object):

def minAddToMakeValid(self, S): ans = bal = 0

for symbol in S:

bal += 1 if symbol == '(' else -1

# It is guaranteed bal >= -1

if bal == -1:

ans += 1

bal += 1

return ans + bal

**92. 922-Sort Array By Parity II**

class Solution(object):

def sortArrayByParityII(self, A):

N = len(A)

ans = [None] \* N

t = 0

for i, x in enumerate(A):

if x % 2 == 0:

ans[t] = x

t += 2

t = 1

for i, x in enumerate(A):

if x % 2 == 1:

ans[t] = x

t += 2

* + #We could have also used slice assignment:
  + #ans[::2] = (x for x in A if x % 2 == 0)

* + #ans[1::2] = (x for x in A if x % 2 == 1)
  + return ans

**93. 925-Long Pressed Name**

class Solution(object):

def isLongPressedName(self, name, typed):

g1 = [(k, len(list(grp))) for k, grp in itertools.groupby(name)]

g2 = [(k, len(list(grp))) for k, grp in itertools.groupby(typed)]

if len(g1) != len(g2):

return False

return all(k1 == k2 and v1 <= v2

for (k1,v1), (k2,v2) in zip(g1, g2))

**94. 932-Beautiful Array**

class Solution:

def beautifulArray(self, N):

memo = {1: [1]}

def f(N):

if N not in memo:

odds = f((N+1)/2)

evens = f(N/2)

memo[N] = [2\*x-1 for x in odds] + [2\*x for x in evens]

return memo[N]

return f(N)

**95. 933-Number of Recent Calls**

class RecentCounter(object):

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

self.q = collections.deque()

def ping(self, t):

self.q.append(t)

while self.q[0] < t-3000:

self.q.popleft()

return len(self.q)

1. **937-Reorder Log Files**

class Solution(object):

def reorderLogFiles(self, logs): def f(log):

id\_, rest = log.split(" ", 1)

return (0, rest, id\_) if rest[0].isalpha() else (1,)

return sorted(logs, key = f)

**97. 939-Minimum Area Rectangle**

class Solution(object):

def minAreaRect(self, points):

columns = collections.defaultdict(list)

for x, y in points:

columns[x].append(y)

lastx = {}

ans = float('inf')

for x in sorted(columns):

column = columns[x]

column.sort()

for j, y2 in enumerate(column):

for i in xrange(j):

y1 = column[i]

if (y1, y2) in lastx:

ans = min(ans, (x - lastx[y1,y2]) \* (y2 - y1))

lastx[y1, y2] = x

return ans if ans < float('inf') else 0

**98. 941-Valid Mountain Array**

class Solution(object):

def validMountainArray(self, A):

N = len(A)

i = 0

# walk up

while i+1 < N and A[i] < A[i+1]:

i += 1

#peak can't be first or last

if i == 0 or i == N-1:

return False

#walk down

while i+1 < N and A[i] > A[i+1]:

i += 1

return i == N-1

**99. 942-DI String Match**

class Solution(object):

def diStringMatch(self, S):

lo, hi = 0, len(S)

ans = []

for x in S:

if x == 'I':

ans.append(lo)

lo += 1

else:

ans.append(hi)

hi -= 1

return ans + [lo]

**100. 944-Delete Columns to Make Sorted**

class Solution(object):

def minDeletionSize(self, A):

ans = 0

for col in zip(\*A):

if any(col[i] > col[i+1] for i in xrange(len(col) - 1)):

ans += 1

return ans

**101. (303)**

class NumArray:

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

self.prefix = [0]

for i in range(len(nums)):

self.prefix.append(self.prefix[i]+nums[i])

def sumRange(self, i: int, j: int) -> int:

return self.prefix[j+1] - self.prefix[i]

102. (307)

class NumArray:

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

self.d=nums

def update(self, i: int, val: int) -> None:

self.d[i]=val

def sumRange(self, i: int, j: int) -> int:

return sum(self.d[i:j+1])

**103. (316)**

class Solution:

def removeDuplicateLetters(self, s) -> int:

stack = []

# this lets us keep track of what's in our solution in O(1) time

seen = set()

# this will let us know if there are no more instances of s[i] left in s

last\_occurrence = {c: i for i, c in enumerate(s)}

for i, c in enumerate(s):

# we can only try to add c if it's not already in our solution

# this is to maintain only one of each character

if c not in seen:

# if the last letter in our solution:

# 1. exists

# 2. is greater than c so removing it will make the string smaller

# 3. it's not the last occurrence

# we remove it from the solution to keep the solution optimal

while stack and c < stack[-1] and i < last\_occurrence[stack[-1]]:

seen.discard(stack.pop())

seen.add(c)

stack.append(c)

return ''.join(stack)

**104. (334)**

<https://leetcode-cn.com/problems/increasing-triplet-subsequence/solution/jian-dan-hao-dong-yi-shi-xian-yi-ci-xun-huan-guo-b/>

class Solution:

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

r1, r2 = sys.maxsize, sys.maxsize

for n in nums :

if n <= r1 : r1 = n

elif n <= r2 : r2 = n

else : return True

return False

**105. (338)**

<https://leetcode-cn.com/problems/counting-bits/solution/338bi-te-wei-ji-shu-by-huster2018/>

class Solution:

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

res = [0]

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

if i%2 == 0:

res.append(res[i//2])

else:

res.append(res[i-1] + 1)

return res

**106. (345)**

class Solution:

def reverseVowels(self, s):

array = ['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U']

sList = list(s)

left, right = 0, len(s)-1

while left < right:

if sList[left] in array and sList[right] in array:

sList[left], sList[right] = sList[right], sList[left]

left += 1

right -= 1

if sList[right] not in array:

right -= 1

if sList[left] not in array:

left += 1

return ''.join(sList)

**107. (371)**

<https://leetcode-cn.com/problems/sum-of-two-integers/solution/wei-yun-suan-xiang-jie-yi-ji-zai-python-zhong-xu-y/>

<https://leetcode-cn.com/problems/sum-of-two-integers/solution/python-wei-yun-suan-yi-xie-keng-by-lih/>

public int getSum(int a, int b) {

if (a == 0) return b;

if (b == 0) return a;

while (b != 0) {

int carry = a & b; //将存在进位的位置置1

a = a ^ b; // 计算无进位的结果

b = carry << 1;

}

return a;

}

（进阶再考虑数据类型的判定问题）

**108. (372)**

<https://leetcode-cn.com/problems/super-pow/solution/you-qian-ru-shen-kuai-su-mi-suan-fa-xiang-jie-by-l/>

<https://leetcode-cn.com/problems/super-pow/solution/kuai-su-mi-qiu-mo-yun-suan-gui-lu-by-desgard_duan/>

class Solution:

def qpow(self, x, n, m):

ans = 1

while n > 0:

if n & 1 == 1:

ans = ans \* x % m

x = x \* x % m

n >>= 1

return ans

def superPow(self, a: int, b: List[int]) -> int:

res = 1

for i in b:

res = self.qpow(res, 10, 1337) \* self.qpow(a, i, 1337)

return res % 1337

**109. (380)**

<https://leetcode-cn.com/problems/insert-delete-getrandom-o1/solution/chang-shu-shi-jian-cha-ru-shan-chu-he-huo-qu-sui-j/>

from random import choice

class RandomizedSet():

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

"""

Initialize your data structure here.

"""

self.dict = {}

self.list = []

def insert(self, val: int) -> bool:

"""

Inserts a value to the set. Returns true if the set did not already contain the specified element.

"""

if val in self.dict:

return False

self.dict[val] = len(self.list)

self.list.append(val)

return True

def remove(self, val: int) -> bool:

"""

Removes a value from the set. Returns true if the set contained the specified element.

"""

if val in self.dict:

# move the last element to the place idx of the element to delete

last\_element, idx = self.list[-1], self.dict[val]

self.list[idx], self.dict[last\_element] = last\_element, idx

# delete the last element

self.list.pop()

del self.dict[val]

return True

return False

def getRandom(self) -> int:

"""

Get a random element from the set.

"""

return choice(self.list)

**110. (384)**

class Solution:

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

self.array = nums

self.original = list(nums)

def reset(self):

self.array = self.original

self.original = list(self.original)

return self.array

def shuffle(self):

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

swap\_idx = random.randrange(i, len(self.array))

self.array[i], self.array[swap\_idx] = self.array[swap\_idx], self.array[i]

return self.array

**111. (392)**

class Solution:

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

n, m = len(s), len(t)

i = j = 0

while i < n and j < m:

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

i += 1

j += 1

return i == n

**112. (412)**

class Solution:

def fizzBuzz(self, n):

"""

:type n: int

:rtype: List[str]

"""

# ans list

ans = []

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

divisible\_by\_3 = (num % 3 == 0)

divisible\_by\_5 = (num % 5 == 0)

if divisible\_by\_3 and divisible\_by\_5:

# Divides by both 3 and 5, add FizzBuzz

ans.append("FizzBuzz")

elif divisible\_by\_3:

# Divides by 3, add Fizz

ans.append("Fizz")

elif divisible\_by\_5:

# Divides by 5, add Buzz

ans.append("Buzz")

else:

# Not divisible by 3 or 5, add the number

ans.append(str(num))

return ans

**113. (415)**

<https://leetcode-cn.com/problems/add-strings/solution/add-strings-shuang-zhi-zhen-fa-by-jyd/>

class Solution:

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

res = ""

i, j, carry = len(num1) - 1, len(num2) - 1, 0

while i >= 0 or j >= 0:

n1 = int(num1[i]) if i >= 0 else 0

n2 = int(num2[j]) if j >= 0 else 0

tmp = n1 + n2 + carry

carry = tmp // 10

res = str(tmp % 10) + res

i, j = i - 1, j - 1

return "1" + res if carry else res

**114. (537)**

class Solution:

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

index\_a, index\_b = 0, 0

for i in range(len(a)):

if a[i] == '+':

index\_a = i

break

for j in range(len(b)):

if b[j] == '+':

index\_b = j

break

a1 = [a[:index\_a], a[index\_a+1:len(a)-1]]

b1 = [b[:index\_b], b[index\_b+1:len(b)-1]]

s1 = int(a1[0]) \* int(b1[0]) - int(a1[1]) \* int(b1[1])

s2 = int(a1[0]) \* int(b1[1]) + int(a1[1]) \* int(b1[0])

return str(s1)+'+'+str(s2)+'i'

**115. (541)**

class Solution:

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

res, flag = "", True

for i in range(0, len(s), k):

res += s[i:i + k][::-1] if flag else s[i:i+k]

flag = not flag

return res

**116. (594)**

class Solution:

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

dicts={}

for i in nums:

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

res=0

for i in dicts:

if i+1 in dicts:

res=max(res,dicts[i]+dicts[i+1])

return res

**117. (599)**

class Solution:

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

d = {x: list1.index(x) + list2.index(x) for x in set(list1) & set(list2)}

return [x for x in d if d[x] == min(d.values())]

**118. (561)**

class Solution:

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

nums.sort()

res = 0

for i in range(len(nums)):

if i % 2 == 0:

res += nums[i]

return res

**119. (538)**

class Solution:

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

def dfs(root: TreeNode):

nonlocal total

if root:

dfs(root.right)

total += root.val

root.val = total

dfs(root.left)

total = 0

dfs(root)

return root

**120. (539)**

class Solution:

def findMinDifference(self, timePoints: List[str]) -> int:

d = set()

for c in timePoints:

k = int(c[: 2]) \* 60 + int(c[3: ])

if k in d: #可能快在了判重这里

return 0

d.add(k)

d = sorted(d)

d.append(d[0] + 1440)

return min(d[i] - d[i - 1] for i in range(1, len(d)))

**121. (540)**

class Solution:

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

return sum(set(nums))\*2 - sum(nums)

**122. (543)**

class Solution:

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

res = 0

def dfs(root):

nonlocal res

if not root: return 0

left = dfs(root.left)

right = dfs(root.right)

res = max(res, left + right)

return 1 + max(left, right)

dfs(root)

return res

**123. (551)**

class Solution:

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

return not( s.count('A') > 1 or "LLL" in s )

**124. (557)**

class Solution:

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

return ' '.join(i[::-1] for i in s.split())

**125. interview 02.03**

# Definition for singly-linked list.

# class ListNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.next = None

class Solution:

    def deleteNode(self, node):

        """

        :type node: ListNode

        :rtype: void Do not return anything, modify node in-place instead.

        """

        node.val = node.next.val

        node.next = node.next.next

**126. 617**

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def mergeTrees(self, t1: TreeNode, t2: TreeNode) -> TreeNode:

        if not t1 or not t2:

            return t1 or t2

        t1.val = t1.val + t2.val

        t1.left = self.mergeTrees(t1.left, t2.left)

        t1.right = self.mergeTrees(t1.right, t2.right)

        return t1

**127. 637**

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def averageOfLevels(self, root: TreeNode):

        res=[]

        queue = [root]

        while queue:

            total\_val = 0

            num = 0

            for \_ in range(len(queue)):

                node = queue.pop(0)

                total\_val += node.val

                num += 1

                if node.left:

                    queue.append(node.left)

                if node.right:

                    queue.append(node.right)

            res.append(total\_val/num)

        return res

**128. 654**

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

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

        if nums == []: return None

        max\_num = max(nums)

        max\_index = nums.index(max\_num)

        root = TreeNode(max\_num)

        root.left = self.constructMaximumBinaryTree(nums[0 : max\_index])

        root.right = self.constructMaximumBinaryTree(nums[max\_index + 1 :])

        return root

**129. 657**

class Solution:

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

        if not moves:

            return True

        U = moves.count("U")

        D = moves.count("D")

        L = moves.count("L")

        R = moves.count("R")

        return U == D and L == R

**130. 700**

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

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

        if not root:

            pass

        else:

            if root.val == val:

                return root

            elif root.val >val:

                return self.searchBST(root.left,val)

            else:

                return self.searchBST(root.right,val)

        return None

**131. 701**

# 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 insertIntoBST(self, root: TreeNode, val: int) -> TreeNode:

        if not root:

            return TreeNode(val)

        def insert(root:TreeNode,val:int):

            if root.val>val and root.left :

                self.insertIntoBST(root.left,val)

            elif root.val > val and not root.left:

                root.left = TreeNode(val)

            elif root.val<val and root.right:

                self.insertIntoBST(root.right,val)

            elif root.val <val and not root.right:

                root.right = TreeNode(val)

        insert(root,val)

        return root

**132. 709**

class Solution:

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

        result = ""

        for s in str:

            if s >= 'A' and s <= 'Z':

                s = chr(ord(s) + 32)

            result += s

        return result

**133. 669**

# 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 trimBST(self, root: TreeNode, low: int, high: int) -> TreeNode:

        if not root:

            return

        if root.val > high:

            return self.trimBST(root.left, low, high)

        elif root.val < low:

            return self.trimBST(root.right, low, high)

        else:

            root.left = self.trimBST(root.left, low, high)

            root.right = self.trimBST(root.right, low, high)

            return root

**134. 682**

class Solution:

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

        res = 0

        temp = []

        for operation in ops:

            if operation == '+':

                res += temp[-1] + temp[-2]

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

            elif operation == 'D':

                res += temp[-1] \* 2

                temp.append(temp[-1] \* 2)

            elif operation == 'C':

                res -= temp[-1]

                temp.pop()

            else:

                res += int(operation)

                temp.append(int(operation))

        return res

**135. 690**

"""

# Definition for Employee.

class Employee:

    def \_\_init\_\_(self, id: int, importance: int, subordinates: List[int]):

        self.id = id

        self.importance = importance

        self.subordinates = subordinates

"""

class Solution:

    def getImportance(self, employees: List['Employee'], id: int) -> int:

        ans = 0

        Emp\_dict = {e.id: e for e in employees}

        boss=[id]

        while boss:

            cur=boss.pop(0)

            e=Emp\_dict[cur]

            ans += e.importance

            boss.extend(e.subordinates)

        return ans

**136. 693**

class Solution:

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

        A = str(bin(n)[2:])

        for i in range(1,len(A)):

            if(A[i] == A[i-1]):return False

        return True

**137. 744**

class Solution:

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

        if letters[len(letters) - 1] <= target:

            return letters[0]

        left = 0

        right = len(letters) - 1

        while left < right:

            mid = (left + right) // 2

            letter = letters[mid]

            if letter <= target:

                left = mid + 1

            else:

                right = mid

        return letters[left]

**138. 747**

class Solution:

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

        ans=nums.index(max(nums))

        temp=nums[ans]

        nums.pop(ans)

        if (nums and temp>=2\*max(nums)) or not nums:

            return ans

        else:

            return -1

**139. 704**

class Solution:

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

        if nums[-1]==target:

            return len(nums)-1

        elif nums[0]==target:

            return 0

        else:

            l=0

            r=len(nums)-1

            while l<r-1:

                m=(l+r)//2

                if nums[m]<target:

                    l=m

                elif nums[m]==target:

                    return m

                else:

                    r=m

        return -1

**140. interview 08.04**

class Solution:

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

        res = [[]]

        if not nums:

            return res

        for num in nums:

            res += [i+[num] for i in res]

        return res

**141. interview 16.01**

class Solution:

    def swapNumbers(self, numbers: List[int]) -> List[int]:

        numbers[0]^=numbers[1]

        numbers[1]^=numbers[0]

        numbers[0]^=numbers[1]

        return numbers

**142. interview 16.07**

class Solution:

    def maximum(self, a: int, b: int) -> int:

        return (abs(a-b)+a+b)//2

**143. Interview 05.06**

why &0xFFFFFFFF in python:

<https://leetcode-cn.com/problems/convert-integer-lcci/solution/yi-huo-jie-fa-python-3-c-by-z1m/>

class Solution:

    def convertInteger(self, A: int, B: int) -> int:

        ans=0

        temp=(A&0xFFFFFFFF)^(B&0xFFFFFFFF)

        st=str(bin(temp)[2:])

        for i in range(len(st)):

            if  st[i]=='1':

                ans+=1

        return ans

**144. Interview 08.03**

class Solution:

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

        for i in range(len(nums)):

            if i==nums[i]:

                return i

        return -1

**145. 1304. Find N Unique Integers Sum up to Zero**

**Solution:**

1. **class** Solution(object):
2. **def** sumZero(self, n):
3. """
4. :type n: int
5. :rtype: List[int]
6. """
7. L = []
8. **for** i **in** range(-(n//2), n//2+1):
9. **if** i != 0:
10. L.append(i)
11. **elif** n & 1:
12. L.append(i)
13. **return** L

#### 146. 1309. Decrypt String from Alphabet to Integer Mapping

**Solution:**

1. **class** Solution(object):
2. **def** freqAlphabets(self, s):
3. """
4. :type s: str
5. :rtype: str
6. """
7. **for** i **in** range(10, 27):
8. s = s.replace("{}#".format(i), chr(ord('a')+i-1))
9. **for** i **in** range(1, 10):
10. s = s.replace("{}".format(i), chr(ord('a')+i-1))
11. **return** s

#### 147. 1310. XOR Queries of a Subarray

**Solution:**

1. **class** Solution(object):
2. **def** xorQueries(self, arr, queries):
3. """
4. :type arr: List[int]
5. :type queries: List[List[int]]
6. :rtype: List[int]
7. """
8. **def** Arr(i):
9. **return** arr[i] **if** i >= 0 **else** 0
11. **for** i **in** range(len(arr)-1):
12. arr[i+1] ^= arr[i]
13. Ans = []
14. **for** query **in** queries:
15. Ans.append(Arr(query[1])^Arr(query[0]-1))
16. **return** Ans

#### 148. 1313. Decompress Run-Length Encoded List

**Solution:**

* **class** Solution(object):
* **def** decompressRLElist(self, nums):
* """
* :type nums: List[int]
* :rtype: List[int]
* """
* Ans = []
* **for** i **in** range(0, len(nums), 2):
* Ans += [nums[i+1]] \* nums[i]
* **return** Ans

#### 149. 1317. Convert Integer to the Sum of Two No-Zero Integers

**Solution:**

1. # By user Thare
2. **class** Solution(object):
3. **def** getNoZeroIntegers(self, n):
4. """
5. :type n: int
6. :rtype: List[int]
7. """
8. **if** n < 10:
9. **return** [1, n-1]
10. A = 0
11. x = n
12. digit = 1
13. **while** x >= 10:
14. delta = (2 **if** x % 10 == 1 **else** 1)
15. A += delta \* digit
16. digit \*= 10
17. x = (x-delta) // 10
18. **return** [A, n-A]

#### 150. 1323. Maximum 69 Number

**Solution:**

1. **class** Solution(object):
2. **def** maximum69Number (self, num):
3. """
4. :type num: int
5. :rtype: int
6. """
7. s = list(str(num))
8. **for** i **in** range(len(s)):
9. **if** s[i] == '6':
10. s[i] = '9'
11. **break**
12. **return** int("".join(s))

#### 151. 1331. Rank Transform of an Array

**Solution:**

1. **class** Solution(object):
2. **def** arrayRankTransform(self, arr):
3. """
4. :type arr: List[int]
5. :rtype: List[int]
6. """
7. arr2 = list(set(arr))
8. arr2.sort()
9. order = dict()
10. **for** i **in** range(len(arr2)):
11. order[arr2[i]] = i
12. **return** [order[i]+1 **for** i **in** arr]

#### 152. 1338. Reduce Array Size to The Half

**Solution:**

1. **class** Solution(object):
2. **def** minSetSize(self, arr):
3. """
4. :type arr: List[int]
5. :rtype: int
6. """
7. count = dict()
8. **for** i **in** arr:
9. **if** i **in** count:
10. count[i] = count[i] + 1
11. **else**:
12. count[i] = 1
13. count = list(count.values())
14. count.sort()
15. remove = 0
16. times = 0
17. **while** remove \* 2 < len(arr):
18. remove += count.pop()
19. times += 1
20. **return** times

#### 153. 1342. Number of Steps to Reduce a Number to Zero

**Solution：**

1. **class** Solution(object):
2. **def** numberOfSteps (self, num):
3. """
4. :type num: int
5. :rtype: int
6. """
7. ans = 0
8. **while** num:
9. **if** num & 1:
10. num -= 1
11. **else**:
12. num >>= 1
13. ans += 1
14. **return** ans

#### 154. 1343. Number of Sub-arrays of Size K and Average Greater than or Equal to Threshold

**Solution：**

1. **class** Solution(object):
2. **def** numOfSubarrays(self, arr, k, threshold):
3. """
4. :type arr: List[int]
5. :type k: int
6. :type threshold: int
7. :rtype: int
8. """
9. **for** i **in** range(len(arr)-1):
10. arr[i+1] += arr[i]
11. **def** Arr(i):
12. **return** arr[i] **if** i >= 0 **else** 0
13. cnt = 0
14. **for** i **in** range(k-1, len(arr)):
15. **if** Arr(i) - Arr(i-k) >= k \* threshold:
16. cnt += 1
17. **return** cnt

#### 155. 1344. Angle Between Hands of a Clock

**Solution:**

1. **class** Solution(object):
2. **def** angleClock(self, hour, minutes):
3. """
4. :type hour: int
5. :type minutes: int
6. :rtype: float
7. """
8. angle\_m = minutes \* 6
9. angle\_h = (hour + minutes / 60.0) \* 30.0
10. **while** angle\_h >= 360:
11. angle\_h -= 360
12. angle = math.fabs(angle\_h - angle\_m)
13. **return** min((angle, 360-angle))

#### 156. 1346. Check If N and Its Double Exist

**Solution:**

1. **class** Solution(object):
2. **def** checkIfExist(self, arr):
3. """
4. :type arr: List[int]
5. :rtype: bool
6. """
7. # using set, O(n) or O(nlog n) depending on set using hash table or balanced tree
8. S = set()
9. **for** i **in** arr:
10. **if** i **in** S:
11. **return** True
12. **if** (i<<2) **in** S:
13. **return** True
14. S.add(i<<1)
15. # avoid floating error
16. **return** False

#### 157. 1356. Sort Integers by The Number of 1 Bits

**Solution:**

1. **class** Solution(object):
2. **def** sortByBits(self, arr):
3. """
4. :type arr: List[int]
5. :rtype: List[int]
6. """
7. **def** bitcount(x):
8. cnt = 0
9. **while** x:
10. cnt += 1
11. x &= (x-1)
12. **return** cnt
13. arr = [(bitcount(i), i) **for** i **in** arr]
14. arr.sort()
15. **return** [i[1] **for** i **in** arr]

#### 158. 1360. Number of Days Between Two Dates

**Solution:**

1. **class** Solution(object):
2. **def** daysBetweenDates(self, date1, date2):
3. """
4. :type date1: str
5. :type date2: str
6. :rtype: int
7. """
8. **def** days(date):
9. y, m, d = date.split('-')
10. y, m, d = int(y), int(m), int(d)
11. ans = (y-1) \* 365 + (y-1) // 4 - (y-1) // 100 + (y-1) // 400
12. months = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
13. **for** i **in** range(m):
14. ans += months[i]
15. ans += (((y % 4 == 0) + (y % 100 == 0) + (y % 400 == 0)) % 2 == 1) **and** (m > 2)
16. ans += d
17. **return** ans
18. **return** int(math.fabs(days(date1) - days(date2)))

#### 159. 1365. How Many Numbers Are Smaller Than the Current Number

**Solution：**

1. **class** Solution(object):
2. **def** smallerNumbersThanCurrent(self, nums):
3. """
4. :type nums: List[int]
5. :rtype: List[int]
6. """
7. nums\_id = [(nums[i], i) **for** i **in** range(len(nums))]
8. nums\_id.sort()
9. ans = [0 **for** i **in** range(len(nums))]
10. **for** i **in** range(len(nums\_id)):
11. **if** i **and** nums\_id[i][0] == nums\_id[i-1][0]:
12. ans[nums\_id[i][1]] = ans[nums\_id[i-1][1]]
13. **else**:
14. ans[nums\_id[i][1]] = i
15. **return** ans

#### 160. 1370. Increasing Decreasing String

**Solution:**

1. **class** Solution(object):
2. **def** sortString(self, s):
3. """
4. :type s: str
5. :rtype: str
6. """
7. cnt = [0 **for** i **in** range(26)]
8. **for** ch **in** s:
9. cnt[ord(ch)-ord('a')] += 1
10. L = []
11. is\_decreasing = False
12. **while** len(L) < len(s):
13. **if** is\_decreasing:
14. **for** i **in** range(25, -1, -1):
15. **if** cnt[i]:
16. cnt[i] -= 1
17. L.append(i)
18. **else**:
19. **for** i **in** range(26):
20. **if** cnt[i]:
21. cnt[i] -= 1
22. L.append(i)
23. is\_decreasing = **not** is\_decreasing
24. **return** "".join([chr(ord('a')+i) **for** i **in** L])

#### 161. 1371. Find the Longest Substring Containing Vowels in Even Counts

**Solution：**

1. **class** Solution(object):
2. **def** findTheLongestSubstring(self, s):
3. """
4. :type s: str
5. :rtype: int
6. """
7. Left = [-2 **for** i **in** range(32)]
8. state = 0
9. Left[state] = -1
10. ans = 0
11. **for** i **in** range(len(s)):
12. state ^= ((s[i] == 'a') << 4) ^ ((s[i] == 'e') << 3) ^ ((s[i] == 'i') << 2) ^ ((s[i] == 'o') << 1) + (s[i] == 'u')
13. **if** Left[state] == -2:
14. Left[state] = i
15. **else**:
16. ans = max(ans, i - Left[state])
17. **return** ans

#### 162. 1380. Lucky Numbers in a Matrix

**Solution:**

1. **class** Solution(object):
2. **def** luckyNumbers (self, matrix):
3. """
4. :type matrix: List[List[int]]
5. :rtype: List[int]
6. """
7. mark = [[0 **for** j **in** range(len(matrix[0]))] **for** i **in** range(len(matrix))]
8. **for** i **in** range(len(matrix)):
9. idx = 0
10. **for** j **in** range(1, len(matrix[0])):
11. **if** matrix[i][j] < matrix[i][idx]:
12. idx = j
13. mark[i][idx] += 1
14. **for** j **in** range(len(matrix[0])):
15. idx = 0
16. **for** i **in** range(1, len(matrix)):
17. **if** matrix[i][j] > matrix[idx][j]:
18. idx = i
19. mark[idx][j] += 1
20. ans = []
21. **for** i **in** range(len(matrix)):
22. **for** j **in** range(len(matrix[0])):
23. **if** mark[i][j] == 2:
24. ans.append(matrix[i][j])
25. **return** ans

#### 163. 1385. Find the Distance Value Between Two Arrays

**Solution:**

1. **class** Solution(object):
2. **def** findTheDistanceValue(self, arr1, arr2, d):
3. """
4. :type arr1: List[int]
5. :type arr2: List[int]
6. :type d: int
7. :rtype: int
8. """
9. arr1.sort()
10. arr2.sort()
11. mark = [1 **for** i **in** range(len(arr1))]
12. i = 0
13. j = 0
14. **while** i < len(arr1) **and** j < len(arr2):
15. **if** arr1[i] <= arr2[j] + d **and** arr1[i] >= arr2[j] - d:
16. mark[i] = 0
17. **if** arr1[i] > arr2[j] + d:
18. j += 1
19. **else**:
20. i += 1
21. cnt = 0
22. **for** i **in** mark:
23. cnt += i
24. **return** cnt

#### 164. 1386. Cinema Seat Allocation

**Solution:**

1. **class** Solution(object):
2. **def** maxNumberOfFamilies(self, n, reservedSeats):
3. """
4. :type n: int
5. :type reservedSeats: List[List[int]]
6. :rtype: int
7. """
8. **def** count(pos):
9. flag = True
10. **for** i **in** range(2, 10):
11. **if** i **in** pos:
12. flag = False
13. **break**
14. **if** flag:
15. **return** 2
16. flag = [True, True, True]
17. **for** i **in** range(0, 4):
18. **if** i+2 **in** pos:
19. flag[0] = False
20. **if** i+4 **in** pos:
21. flag[1] = False
22. **if** i+6 **in** pos:
23. flag[2] = False
24. **if** True **in** flag:
25. **return** 1
26. **return** 0
27. reservedSeats.sort()
28. pos = []
29. ans = 2 \* (reservedSeats[0][0] - 1)
30. **for** i **in** range(len(reservedSeats)):
31. **if** i **and** reservedSeats[i][0] != reservedSeats[i-1][0]:
32. ans += count(pos)
33. ans += 2 \* (reservedSeats[i][0] - reservedSeats[i-1][0] - 1)
34. pos = []
35. pos.append(reservedSeats[i][1])
36. **return** ans + count(pos) + 2 \* (n - reservedSeats[-1][0])

#### 165. 1394. Find Lucky Integer in an Array

**Solution:**

1. **class** Solution(object):
2. **def** findLucky(self, arr):
3. """
4. :type arr: List[int]
5. :rtype: int
6. """
7. cnt = [0 **for** i **in** range(510)]
8. **for** i **in** arr:
9. cnt[i] += 1
10. ans = -1
11. **for** i **in** arr:
12. **if** cnt[i] == i:
13. ans = max((ans, i))
14. **return** ans

#### 166. 1396. Design Underground System

**Solution:**

1. **class** UndergroundSystem(object):
3. **def** \_\_init\_\_(self):
4. self.count = dict()
5. self.totalTime = dict()
6. self.start = dict()
8. **def** checkIn(self, id, stationName, t):
9. """
10. :type id: int
11. :type stationName: str
12. :type t: int
13. :rtype: None
14. """
15. self.start[id] = (stationName, t)

18. **def** checkOut(self, id, stationName, t):
19. """
20. :type id: int
21. :type stationName: str
22. :type t: int
23. :rtype: None
24. """
25. startStation, startTime = self.start[id]
26. **if** (startStation, stationName) **in** self.count:
27. self.count[(startStation, stationName)] += 1
28. self.totalTime[(startStation, stationName)] += t - startTime
29. **else**:
30. self.count[(startStation, stationName)] = 1
31. self.totalTime[(startStation, stationName)] = t - startTime
33. **def** getAverageTime(self, startStation, endStation):
34. """
35. :type startStation: str
36. :type endStation: str
37. :rtype: float
38. """
39. **return** 1.0 \* self.totalTime[(startStation, endStation)] / self.count[(startStation, endStation)]


43. # Your UndergroundSystem object will be instantiated and called as such:
44. # obj = UndergroundSystem()
45. # obj.checkIn(id,stationName,t)
46. # obj.checkOut(id,stationName,t)
47. # param\_3 = obj.getAverageTime(startStation,endStation)

**167. 1403. Minimum Subsequence in Non-Increasing Order**

1. **class** Solution:
2. **def** minSubsequence(self, nums: List[int]) -> List[int]:
3. total = sum(nums)
4. res = []
5. cur\_total = 0
6. **for** n **in** sorted(nums)[::-1]:
7. res.append(n)
8. cur\_total += n
9. **if** cur\_total > total/2:
10. **return** res

**168. 1408. String Matching in an Array**

* **class** Solution:
* **def** stringMatching(self, words: List[str]) -> List[str]:
* res = []
* sorted\_words = sorted(words, key=**lambda** x: len(x))
* **for** i **in** range(len(sorted\_words)):
* **for** j **in** range(i+1,len(sorted\_words)):
* **if** sorted\_words[i] **in** sorted\_words[j]:
* res.append(sorted\_words[i])
* **break**
* **return** res

**169. 1413. Minimum Value to Get Positive Step by Step Sum**

* **class** Solution:
* **def** minStartValue(self, nums: List[int]) -> int:
* res = 1
* add = 0
* cur\_value = 1
* **for** n **in** nums:
* cur\_value += n
* **if** cur\_value < 1:
* add += 1-cur\_value
* cur\_value = 1
* **return** res+add

**170. 1417. Reformat The String**

1. **class** Solution:
2. **def** reformat(self, s: str) -> str:
3. letters = [c **for** c **in** s **if** c >= 'a']
4. digits = [c **for** c **in** s **if** c < 'a']
5. l1 = len(letters)
6. l2 = len(digits)
7. **if** abs(l1-l2) > 1:
8. **return** ''
9. **elif** l1 > l2:
10. res = ''
11. **for** i **in** range(len(digits)):
12. res+=letters[i]
13. res+=digits[i]
14. res+=letters[-1]
15. **elif** l2 > l1:
16. res = ''
17. **for** i **in** range(len(letters)):
18. res+=digits[i]
19. res+=letters[i]
20. res+=digits[-1]
21. **else**:
22. res = ''
23. **for** i **in** range(len(digits)):
24. res+=digits[i]
25. res+=letters[i]
27. **return** res

**171. 1422. Maximum Score After Splitting a String**

1. **class** Solution:
2. **def** maxScore(self, s: str) -> int:
3. l = [int(c) **for** c **in** s]
4. res = []
5. **for** i **in** range(1,len(l)):
6. res.append(i-sum(l[:i])+sum(l[i:]))
7. **return** max(res)

**172. 1431. Kids With the Greatest Number of Candies**

1. **class** Solution:
2. **def** kidsWithCandies(self, candies: List[int], extraCandies: int) -> List[bool]:
3. m = max(candies)
4. **return** [(candies[i]+extraCandies)>=m **for** i **in** range(len(candies))]

**173. 1441. Build an Array With Stack Operations**

1. **class** Solution:
2. **def** buildArray(self, target: List[int], n: int) -> List[str]:
3. res = []
4. cur = []
5. **for** i **in** range(1,n+1):
6. **if** i **in** target:
7. res.append('Push')
8. cur.append(i)
9. **else**:
10. res.append('Push')
11. res.append('Pop')
12. **if** cur == target:
13. **return** res

**174. 1446. Consecutive Characters**

1. **class** Solution:
2. **def** maxPower(self, s: str) -> int:
3. res = 1
4. cur = 1
5. **for** i **in** range(1,len(s)):
6. **if** s[i] == s[i-1]:
7. cur += 1
8. **else**:
9. res = max(res,cur)
10. cur = 1
11. **return** max(res,cur)

**175. 1450. Number of Students Doing Homework at a Given Time**

1. **class** Solution:
2. **def** busyStudent(self, startTime: List[int], endTime: List[int], queryTime: int) -> int:
3. **return** sum(startTime[i]<=queryTime<=endTime[i] **for** i **in** range(len(startTime)))

**176. 1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence**

1. **class** Solution:
2. **def** isPrefixOfWord(self, sentence: str, searchWord: str) -> int:
3. l = sentence.split(' ')
4. **for** i **in** range(len(l)):
5. **if** searchWord[0] == l[i][0] **and** searchWord **in** l[i]:
6. **return** i+1
7. **return** -1

**177. 1460. Make Two Arrays Equal by Reversing Sub-arrays**

1. **class** Solution:
2. **def** canBeEqual(self, target: List[int], arr: List[int]) -> bool:
3. dic1 = collections.Counter(target)
4. dic2 = collections.Counter(arr)
5. **return** dic1 == dic2

**178. 1464. Maximum Product of Two Elements in an Array**

1. **class** Solution:
2. **def** maxProduct(self, nums: List[int]) -> int:
3. products = []
4. **for** i **in** range(len(nums)):
5. **for** j **in** range(i+1,len(nums)):
6. products.append((nums[i]-1)\*(nums[j]-1))
7. **return** max(products)

**179. 1470. Shuffle the Array**

1. **class** Solution:
2. **def** shuffle(self, nums: List[int], n: int) -> List[int]:
3. res = []
4. **for** i **in** range(n):
5. res.append(nums[i])
6. res.append(nums[n+i])
7. **return** res

**180. 1475. Final Prices With a Special Discount in a Shop**

1. **class** Solution:
2. **def** finalPrices(self, prices: List[int]) -> List[int]:
3. stack = []
4. discount = [-1]\*len(prices)
5. **for** i **in** range(len(prices)):
6. **while** stack **and** prices[stack[-1]] >= prices[i]:
7. discount[stack.pop()] = i
8. stack.append(i)
9. res = []
10. **for** i **in** range(len(prices)):
11. **if** discount[i] != -1:
12. res.append(prices[i]-prices[discount[i]])
13. **else**:
14. res.append(prices[i])
15. **return** res

**181. 1480. Running Sum of 1d Array**

1. **class** Solution:
2. **def** runningSum(self, nums: List[int]) -> List[int]:
3. **return** [sum(nums[:i]) **for** i **in** range(1,len(nums)+1)]

**182. 1486. XOR Operation in an Array**

1. **class** Solution:
2. **def** xorOperation(self, n: int, start: int) -> int:
3. **return** reduce(xor,[start+2\*i **for** i **in** range(n)])

**183. 1491. Average Salary Excluding the Minimum and Maximum Salary**

1. **class** Solution:
2. **def** average(self, salary: List[int]) -> float:
3. **return** (sum(salary)-max(salary)-min(salary))/(len(salary)-2)

**184. 1496. Path Crossing**

1. **class** Solution:
2. **def** isPathCrossing(self, path: str) -> bool:
3. start = [0,0]
4. passed = set()
5. passed.add((0,0))
6. **for** s **in** path:
7. **if** s == 'N':
8. start[1] += 1
9. **elif** s == 'S':
10. start[1] += -1
11. **elif** s == 'E':
12. start[0] += 1
13. **else**:
14. start[0] += -1
15. **if** tuple(start) **in** passed:
16. **return** True
17. **else**:
18. passed.add(tuple(start))
19. **return** False

**185. 1502. Can Make Arithmetic Progression From Sequence**

1. **class** Solution:
2. **def** canMakeArithmeticProgression(self, arr: List[int]) -> bool:
3. **if** len(arr) == 2:
4. **return** True
5. sorted\_arr = sorted(arr)
6. d = sorted\_arr[1]-sorted\_arr[0]
7. **for** i **in** range(2,len(arr)):
8. **if** sorted\_arr[i]-sorted\_arr[i-1] != d:
9. **return** False
10. **return** True

**186. 1512. Number of Good Pairs**

1. **class** Solution:
2. **def** numIdenticalPairs(self, nums: List[int]) -> int:
3. res = 0
4. **for** i **in** range(len(nums)):
5. **for** j **in** range(i+1,len(nums)):
6. **if** nums[i] == nums[j]:
7. res += 1
8. **return** res

**187. 1518. Water Bottles**

1. **class** Solution:
2. **def** numWaterBottles(self, numBottles: int, numExchange: int) -> int:
3. drink = numBottles
4. empty = numBottles
5. **while** empty >= numExchange:
6. empty -= numExchange
7. drink += 1
8. empty += 1
9. **return** drink

**188. 1523. Count Odd Numbers in an Interval Range**

1. **class** Solution:
2. **def** countOdds(self, low: int, high: int) -> int:
3. **return** (high-low)//2+1 **if** (low%2 **or** high%2) **else** (high-low)//2

**189. 1528. Shuffle String**

1. **class** Solution:
2. **def** restoreString(self, s: str, indices: List[int]) -> str:
3. res = ['' **for** i **in** range(len(s))]
4. **for** i **in** range(len(s)):
5. res[indices[i]] += s[i]
6. **return** ''.join(res)

**190. 1534. Count Good Triplets**

1. **class** Solution:
2. **def** countGoodTriplets(self, arr: List[int], a: int, b: int, c: int) -> int:
3. total = 0
4. **for** i **in** range(len(arr)):
5. **for** j **in** range(i+1,len(arr)):
6. **for** k **in** range(j+1,len(arr)):
7. **if** abs(arr[i] - arr[j])<=a **and** abs(arr[j] - arr[k])<=b **and** abs(arr[i] - arr[k])<=c:
8. total += 1
10. **return** total

**191. 1539. Kth Missing Positive Number**

1. **class** Solution:
2. **def** findKthPositive(self, arr: List[int], k: int) -> int:
3. count = 0
4. **for** i **in** range(1,k+len(arr)+1):
5. **if** i **not** **in** arr:
6. count += 1
7. **if** count == k:
8. **return** i

**192. 1550. Three Consecutive Odds**

1. **class** Solution:
2. **def** threeConsecutiveOdds(self, arr: List[int]) -> bool:
3. consecutive\_count = 0
4. **for** a **in** arr:
5. **if** a % 2:
6. consecutive\_count += 1
7. **else**:
8. consecutive\_count = 0
9. **if** consecutive\_count == 3:
10. **return** True
11. **return** False

193. 766. Toeplitz Matrix

class Solution:

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

for r in range(len(matrix)):

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

if r > 0 and c > 0 and matrix[r-1][c-1] != matrix[r][c]:

return False

return True

194. 771. Jewels and Stones

class Solution:

def numJewelsInStones(self, J: str, S: str) -> int:

set\_j = set()

for ch in J:

set\_j.add(ch)

ans = 0

for ch in S:

if ch in set\_j:

ans += 1

return ans

195. 796. Rotate String

class Solution:

def rotateString(self, A: str, B: str) -> bool:

return len(A) == len(B) and B in A+A

196. 804. Unique Morse Code Words

class Solution:

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

M = (".-", "-...", "-.-.", "-..", ".", "..-.",

"--.", "....", "..", ".---", "-.-", ".-..",

"--", "-.", "---", ".--.", "--.-", ".-.", "...",

"-", "..-", "...-", ".--", "-..-", "-.--", "--..")

l = set()

for i in words:

st = self.MorseString(i, M)

if st not in l:

l.add(st)

return len(l)

def MorseString(self, s, M):

ss = ""

for i in s:

ss += M[ord(i) - ord('a')]

return ss

197. 811. Subdomain Visit Count

class Solution:

def subdomainVisits(self, cpdomains: List[str]) -> List[str]:

ans = []

d1 = {}

for i in range(len(cpdomains)):

input\_ = cpdomains[i].split(' ')

self.add\_dict(input\_[0], input\_[1], d1)

# Each address will have either 1 or 2 "." characters.

i1 = input\_[1].find('.')

self.add\_dict(input\_[0], input\_[1][i1+1:], d1)

i2 = input\_[1][i1+1:].find('.')

if i2 != -1:

self.add\_dict(input\_[0], input\_[1][i1+i2+2:], d1)

for i in d1:

ans.append(str(d1[i]) + ' ' + i)

return ans

def add\_dict(self, s1, s2, d1):

if s2 in d1:

d1[s2] += int(s1)

else:

d1[s2] = int(s1)

198. 819. Most Common Word

class Solution:

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

st = ""

d = {}

s\_banned = set(banned)

ig = (" ", "?", "!", "'", ",", ";", ".")

for ch in paragraph:

ch = ch.lower()

if ch in ig:

if st and st not in s\_banned:

if st in d:

d[st] += 1

else:

d[st] = 1

st = ""

else:

st += ch

if st and st not in s\_banned:

if st in d:

d[st] += 1

else:

d[st] = 1

return max(d, key=d.get)

199. 821. Shortest Distance to a Character

class Solution:

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

prev = -100000

ans = [0] \* len(S)

for i, ch in enumerate(S):

if ch == C:

prev = i

ans[i] = i - prev

prev = 100000

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

if S[i] == C:

prev = i

ans[i] = min(ans[i], prev - i)

return ans

200. 824. Goat Latin

class Solution:

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

l = S.split(' ')

ans = ''

i = 1

for s in l:

ans += (self.wordToGoatLatin(s) + 'a' \* i)

ans += ' '

i += 1

return ans[:-1]

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

if s[0] in ('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'):

return s + 'ma'

else:

return s[1:] + s[0] + 'ma'

201. 830. Positions of Large Groups

class Solution:

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

ans = []

i = 0 # The start of each group

for j in range(len(s)):

if j == len(s) - 1 or s[j] != s[j+1]:

# Here, [i, j] represents a group.

if j-i+1 >= 3:

ans.append([i, j])

i = j+1

return ans

202. 836. Rectangle Overlap

class Solution:

def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:

return not (rec1[2] <= rec2[0] or # left

rec1[3] <= rec2[1] or # bottom

rec1[0] >= rec2[2] or # right

rec1[1] >= rec2[3]) # top

203. 844. Backspace String Compare

class Solution:

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

def build(S):

ans = []

for c in S:

if c != '#':

ans.append(c)

elif ans:

ans.pop()

return "".join(ans)

return build(S) == build(T)

204. 852. Peak Index in a Mountain Array

class Solution:

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

lo, hi = 0, len(arr) - 1

while lo < hi:

mi = (lo + hi) // 2

if arr[mi] < arr[mi + 1]:

lo = mi + 1

else:

hi = mi

return lo

205. 860. Lemonade Change

class Solution:

def lemonadeChange(self, bills: List[int]) -> bool:

five = ten = 0

for bill in bills:

if bill == 5:

five += 1

elif bill == 10:

if not five:

return False

five -= 1

ten += 1

else:

if ten and five:

ten -= 1

five -= 1

elif five >= 3:

five -= 3

else:

return False

return True

206. 867. Transpose Matrix

class Solution:

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

res = []

for i in range(len(A[0])):

temp = []

for j in range(len(A)):

temp.append(A[j][i])

res.append(temp)

return res

207. 868. Binary Gap

class Solution:

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

last = None

ans = 0

for i in range(32):

if n % 2 == 1:

if last is not None:

ans = max(ans, i - last)

last = i

n >>= 1

return ans

208. 876. Middle of the Linked List

# Definition for singly-linked list.

# class ListNode:

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

# self.val = val

# self.next = next

class Solution:

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

slow = fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

return slow

209. 884. Uncommon Words from Two Sentences

class Solution:

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

count = {}

for word in A.split():

count[word] = count.get(word, 0) + 1

for word in B.split():

count[word] = count.get(word, 0) + 1

return [word for word in count if count[word] == 1]

210. 896. Monotonic Array

class Solution:

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

increasing = decreasing = True

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

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

increasing = False

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

decreasing = False

return increasing or decreasing

211. 905. Sort Array By Parity

class Solution:

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

i, j = 0, len(A) - 1

while i < j:

if A[i] % 2 > A[j] % 2:

A[i], A[j] = A[j], A[i]

if A[i] % 2 == 0:

i += 1

if A[j] % 2 == 1:

j -= 1

return A

212. 917. Reverse Only Letters

class Solution:

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

ch\_list = [ch for ch in S]

i = 0

j = len(ch\_list) - 1

while i < j:

while i < j and not ch\_list[i].isalpha():

i += 1

while i < j and not ch\_list[j].isalpha():

j -= 1

if i < j:

ch\_list[i], ch\_list[j] = ch\_list[j], ch\_list[i]

i += 1

j -= 1

return ''.join(ch\_list)

213. 922. Sort Array By Parity II

class Solution:

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

j = 1

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

if A[i] % 2:

while A[j] % 2:

j += 2

A[i], A[j] = A[j], A[i]

return A

214. 925. Long Pressed Name

class Solution:

def isLongPressedName(self, name: str, typed: str) -> bool:

i = 0

for j in range(len(typed)):

if i < len(name) and name[i] == typed[j]:

i += 1

elif j == 0 or typed[j] != typed[j - 1]:

return False

return i == len(name)

215. 929. Unique Email Addresses

class Solution:

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

seen = set()

for email in emails:

local, domain = email.split('@')

if '+' in local:

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

seen.add(local.replace('.','') + '@' + domain)

return len(seen)

216. 941. Valid Mountain Array

class Solution:

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

N = len(A)

i = 0

# walk up

while i+1 < N and A[i] < A[i+1]:

i += 1

# peak can't be first or last

if i == 0 or i == N-1:

return False

# walk down

while i+1 < N and A[i] > A[i+1]:

i += 1

return i == N-1

217. 942. DI String Match

class Solution:

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

lo, hi = 0, len(S)

ans = []

for x in S:

if x == 'I':

ans.append(lo)

lo += 1

else:

ans.append(hi)

hi -= 1

return ans + [lo]

218. 953. Verifying an Alien Dictionary

class Solution:

def isAlienSorted(self, words: List[str], order: str) -> bool:

order\_dict = {order[i]: i for i in range(len(order))}

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

if not self.comp(words[i], words[i+1], order\_dict):

return False

return True

def comp(self, w1, w2, order\_dict):

for ch1, ch2 in zip(w1, w2):

if order\_dict[ch1] < order\_dict[ch2]:

return True # w1 < w2

elif order\_dict[ch1] > order\_dict[ch2]:

return False # w1 > w2

return len(w1) <= len(w2)

219. 977. Squares of a Sorted Array

class Solution:

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

return sorted(x\*x for x in A)

220. 985. Sum of Even Numbers After Queries

class Solution:

def sumEvenAfterQueries(self, A: List[int], queries: List[List[int]]) -> List[int]:

S = sum(x for x in A if x % 2 == 0)

ans = []

for x, k in queries:

if A[k] % 2 == 0: S -= A[k]

A[k] += x

if A[k] % 2 == 0: S += A[k]

ans.append(S)

return ans

221. 989. Add to Array-Form of Integer

class Solution:

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

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

if K == 0:

break

A[i] += K

K = A[i] // 10

A[i] %= 10

while K > 0:

A.insert(0, K % 10)

K //= 10

return A

**222. 1002. Find Common Characters**

class Solution:

def commonChars(self, A):

if not A:

return []

res = []

for c in set(A[0]):

count = [w.count(c) for w in A]

s = c \* min(count) # 如果不是每个单词都有的字母，min(count)=0

for a in s:

res.append(a)

return res

**223. 1005. Maximize Sum Of Array After K Negations**

class Solution:

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

        A.sort()

        i = 0

        while K > 0 and i < len(A):

            if A[i] < 0:

                A[i] = -A[i]

                K -= 1

                i += 1

            else:

                break

        if K % 2 == 0:

            return sum(A)

        else:

            A.sort()

            A[0] = -A[0]

            return sum(A)

**224. 1009. Complement of Base 10 Integer**

class Solution:

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

Nbin=bin(N)

return 2\*\*len(Nbin[2:])-1-N

**225. 1010. Pairs of Songs With Total Durations Divisible by 60**

class Solution:

    def numPairsDivisibleBy60(self, time: List[int]) -> int:

        dic = {}

        for i in range(len(time)):

            if time[i] % 60 in dic:

                dic[time[i] % 60] += 1

            else:

                dic[time[i] % 60] = 1

        res = 0

        for k in dic:

            if k == 0 or k == 30:

                n = dic[k]

                res += n \* (n - 1) // 2

            elif k < 30 and 60 - k in dic:

                res += dic[k] \* dic[60 - k]

        return res

**226. 1013. Partition Array Into Three Parts With Equal Sum**

class Solution:

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

s = sum(A)

if s % 3 != 0:

return False

target = s // 3

n, i, cur = len(A), 0, 0

while i < n:

cur += A[i]

if cur == target:

break

i += 1

if cur != target:

return False

j = i + 1

while j + 1 < n: # 需要满足最后一个数组非空

cur += A[j]

if cur == target \* 2:

return True

j += 1

return False

**227. 1018. Binary Prefix Divisible By 5**

class Solution:

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

        res = []

        num = 0

        for i in range(len(A)):

            num = (num \* 2 + A[i]) % 5

            res.append(num == 0)

        return res

**228. 1025. Divisor Game**

class Solution:

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

        return N % 2 == 0

**229. 1030. Matrix Cells in Distance Order**

class Solution:

    def allCellsDistOrder(self, R, C, r0, c0):

        dist\_list = [[] for i in range(200)]

        for i in range(R):

            for j in range(C):

                distinct = abs(r0-i) + abs(c0-j)

                dist\_list[distinct].append([i, j])

        result = []

        for i in dist\_list:

            if i:

                result.extend(i)

        return result

**230. 1033. Moving Stones Until Consecutive**

class Solution:

def numMovesStones(self, a: int, b: int, c: int) -> List[int]:

x,y,z=sorted([a,b,c])

if x+1==y and y+1==z: #三连续

return [0,0]

if x+1<y and y+1==z: #右连续

return [1,y-x-1]

if x+1==y and y+1<z: #左连续

return [1,z-y-1]

if x+2==y or y+2==z: #左边两数相差2，或者右边两数相差2

return [1,z-x-2]

if x+1<y and y+1<z:

return [2,z-x-2]

**231. 1037. Valid Boomerang**

class Solution:

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

if (points[1][0]==points[0][0]):

k1=float('inf')

else:

k1=(points[1][1]-points[0][1])/(points[1][0]-points[0][0])

if (points[2][0]==points[1][0]):

k2=float('inf')

else:

k2=(points[2][1]-points[1][1])/(points[2][0]-points[1][0])

if (points[2][0]==points[0][0]):

k3=float('inf')

else:

k3=(points[2][1]-points[0][1])/(points[2][0]-points[0][0])

if k1==k2 or k1==k3 or k2==k3:

return False

else:

return True

**232. 1047. Remove All Adjacent Duplicates In String**

class Solution:

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

output = []

for ch in S:

if output and ch == output[-1]:

output.pop()

else:

output.append(ch)

return ''.join(output)

**233. 1051. Height Checker**

class Solution:

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

        h\_sort = sorted(heights)

        res = 0

        for i in range(len(heights)):

            if heights[i] != h\_sort[i]:

                res += 1

        return res

**234. 1078. Occurrences After Bigram**

class Solution:

def findOcurrences(self, text: str, first: str, second: str) -> List[str]:

word\_lst = text.split()

res = []

for i in range(1,len(word\_lst)):

if word\_lst[i-1] == first and word\_lst[i] == second and i+1<len(word\_lst):

res.append(word\_lst[i+1])

return res

**235. 1103. Distribute Candies to People**

class Solution:

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

ans = [0] \* num\_people

i = 0

while candies != 0:

ans[i % num\_people] += min(i + 1, candies)

candies -= min(i + 1, candies)

i += 1

return ans

**236. 1108. Defanging an IP Address**

class Solution:

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

return address.replace('.','[.]')

**237. 1122. Relative Sort Array**

class Solution:

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

arr = [0 for \_ in range(1001)]

ans = [] 。

for i in range(len(arr1)):

arr[arr1[i]] += 1

for i in range(len(arr2)):

while arr[arr2[i]] > 0:

ans.append(arr2[i])

arr[arr2[i]] -= 1

for i in range(len(arr)):

while arr[i] > 0:

ans.append(i)

arr[i] -= 1

return ans # 返回最终答案。

**238. 1137. N-th Tribonacci Number**

class Solution:

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

T =[0,1,1]

for i in range(n-2):

T.append(T[i]+T[i+1]+T[i+2])

return T[n]

239. **[1556. Thousand Separator](https://leetcode-cn.com/problems/thousand-separator/)**

1. **class** Solution:
2. **def** thousandSeparator(self, n: int) -> str:
3. n, ans = list(str(n)), ""
4. length = len(n)
5. **for** i **in** range(1, length + 1):
6. ans += n[length - i]
7. **if** i % 3 == 0:
8. ans += '.'
9. **return** ans[:-1][::-1] **if** length % 3 == 0 **else** ans[::-1]

240. **[1560. Most Visited Sector in a Circular Track](https://leetcode-cn.com/problems/most-visited-sector-in-a-circular-track/)**

1. **class** Solution:
2. **def** mostVisited(self, n: int, rounds: List[int]) -> List[int]:
3. **if** **not** rounds:
4. **return** []
5. a, b = rounds[0], rounds[-1]
6. **return** list(range(a, b + 1)) **if** a <= b **else** list(range(1, b + 1)) + list(range(a, n + 1))

241. **[1572. Matrix Diagonal Sum](https://leetcode-cn.com/problems/matrix-diagonal-sum/)**

1. **class** Solution:
2. **def** diagonalSum(self, mat: List[List[int]]) -> int:
3. n = len(mat)
4. **return** sum(mat[i][i] **for** i **in** range(n)) + \
5. sum(mat[i][n - i - 1] **for** i **in** range(n)) - \
6. (mat[n - 1 >> 1][n - 1 >> 1] **if** (n & 1) **else** 0)

242. **[1573. Number of Ways to Split a String](https://leetcode-cn.com/problems/number-of-ways-to-split-a-string/)**

1. **class** Solution:
2. **def** numWays(self, s: str) -> int:
3. Sum = sum(1 **if** x == '1' **else** 0 **for** x **in** s)
4. **if** Sum % 3:
5. **return** 0
6. **if** Sum == 0:
7. **return** ((len(s) - 2) \* (len(s) - 1) >> 1) % int(1e9 + 7)
8. Sum /= 3
9. Sum2 = Sum + Sum
10. t1, t2, now = 0, 0, 0
11. **for** x **in** range(len(s)):
12. now += (s[x] == '1')
13. t1 += (now == Sum)
14. t2 += (now == Sum2)
15. **return** t1 \* t2 % int(1e9 + 7)

243. **[1576. Replace All ?'s to Avoid Consecutive Repeating Characters](https://leetcode-cn.com/problems/replace-all-s-to-avoid-consecutive-repeating-characters/)**

1. **class** Solution:
2. **def** modifyString(self, s: str) -> str:
3. **def** get\_next(x, ban):
4. **if** x == '?':
5. **return** 'a'
6. x = chr(ord(x) + 1) **if** x != 'z' **else** 'a'
7. **if** x == ban:
8. x = chr(ord(x) + 1) **if** x != 'z' **else** 'a'
9. **return** x
10. **if** s == '?':
11. **return** 'a'
12. ans = []
13. **for** t **in** range(len(s)):
14. ans.append(
15. s[t] **if** s[t] != '?' **else** (
16. get\_next(ans[-1], s[t + 1]) **if** t > 0 **and** t < len(s) - 1 **else** (
17. get\_next(ans[-1], -1) **if** t > 0 **else** get\_next(s[t + 1], -1)
18. )
19. )
20. )
21. **return** ''.join(ans)

244. **[1578. Minimum Deletion Cost to Avoid Repeating Letters](https://leetcode-cn.com/problems/minimum-deletion-cost-to-avoid-repeating-letters/)**

1. **class** Solution:
2. **def** minCost(self, s: str, cost: List[int]) -> int:
3. ans, now\_character, now\_sum, now\_max = 0, s[0], cost[0], cost[0]
4. **for** x **in** range(1, len(s)):
5. **if** s[x] == now\_character:
6. now\_sum += cost[x]
7. now\_max = max(now\_max, cost[x])
8. **else**:
9. ans += now\_sum - now\_max
10. now\_character = s[x]
11. now\_sum = now\_max = cost[x]
12. **return** ans + now\_sum - now\_max

245. **[1582. Special Positions in a Binary Matrix](https://leetcode-cn.com/problems/special-positions-in-a-binary-matrix/)**

1. **class** Solution:
2. **def** numSpecial(self, mat: List[List[int]]) -> int:
3. **import** numpy as np
4. mat = np.array(mat)
5. sum1, sum2 = mat.sum(0), mat.sum(1)
6. xpos, ypos = np.where(mat > 0)
7. ans = 0
8. **for** t **in** range(len(xpos)):
9. x, y = xpos[t], ypos[t]
10. ans += (sum1[y] == 1 **and** sum2[x] == 1)
11. **return** int(ans)

246. **[1592. Rearrange Spaces Between Words](https://leetcode-cn.com/problems/rearrange-spaces-between-words/)**

1. **class** Solution:
2. **def** reorderSpaces(self, text: str) -> str:
3. spacenum = sum(1 **for** x **in** text **if** x == ' ')
4. text\_split = text.split()
5. word\_count = len(text\_split)
6. **if** word\_count == 1:
7. **return** text\_split[0] + spacenum \* ' '
8. div, mod = divmod(spacenum, word\_count - 1)
9. **return** (' ' \* div).join(text\_split) + mod \* ' '

247. **[1598. Crawler Log Folder](https://leetcode-cn.com/problems/crawler-log-folder/)**

1. **class** Solution:
2. **def** minOperations(self, logs: List[str]) -> int:
3. depth = 0
4. **for** x **in** logs:
5. depth += max(
6. -depth, -1 **if** x == '../' **else**(
7. 0 **if** x == './' **else** 1
8. )
9. )
10. **return** depth

248. **[1603. Design Parking System](https://leetcode-cn.com/problems/design-parking-system/)**

1. **class** ParkingSystem:
3. **def** \_\_init\_\_(self, big: int, medium: int, small: int):
4. self.Res = [0, big, medium, small]
6. **def** addCar(self, carType: int) -> bool:
7. **if** self.Res[carType]:
8. self.Res[carType] -= 1
9. **return** True
10. **else**:
11. **return** False

249. **[1608. Special Array With X Elements Greater Than or Equal X](https://leetcode-cn.com/problems/special-array-with-x-elements-greater-than-or-equal-x/)**

1. **class** Solution:
2. **def** specialArray(self, nums: List[int]) -> int:
3. n = len(nums)
4. nums.sort()
5. **if** n <= nums[0]:
6. **return** n
7. **for** i **in** range(1, n):
8. **if** nums[i] >= n - i **and** nums[i - 1] < n - i:
9. **return** n - i
10. **return** -1

250. **[1609. Even Odd Tree](https://leetcode-cn.com/problems/even-odd-tree/)**

1. # Definition for a binary tree node.
2. # class TreeNode:
3. #     def \_\_init\_\_(self, val=0, left=None, right=None):
4. #         self.val = val
5. #         self.left = left
6. #         self.right = right
7. **class** Solution:
8. **def** isEvenOddTree(self, root: TreeNode) -> bool:
10. **def** dfs(x, level, level\_nodes):
11. **if** **not** x:
12. **return**
13. dfs(x.left, level + 1, level\_nodes)
14. **if** level **in** level\_nodes:
15. level\_nodes[level].append(x.val)
16. **else**:
17. level\_nodes[level] = [x.val]
18. dfs(x.right, level + 1, level\_nodes)
20. level\_To\_val = {}
21. dfs(root, 0, level\_To\_val)
22. **for** k, v **in** level\_To\_val.items():
23. last = None
24. **for** x **in** v:
25. **if** (x & 1) == (k & 1):
26. **return** False
27. **for** x **in** range(1, len(v)):
28. **if** (k & 1) **and** v[x] >= v[x - 1]:
29. **return** False
30. **elif** **not** (k & 1) **and** v[x] <= v[x - 1]:
31. **return** False
32. **return** True

251. **[1614. Maximum Nesting Depth of the Parentheses](https://leetcode-cn.com/problems/maximum-nesting-depth-of-the-parentheses/)**

1. **class** Solution:
2. **def** maxDepth(self, s: str) -> int:
3. now\_depth, max\_depth = 0, 0
4. **for** x **in** s:
5. **if** x == '(':
6. now\_depth += 1
7. **if** x == ')':
8. max\_depth = max(max\_depth, now\_depth)
9. now\_depth -= 1
10. **return** max\_depth - now\_depth

252. **[1615. Maximal Network Rank](https://leetcode-cn.com/problems/maximal-network-rank/)**

1. **class** Solution:
2. **def** maximalNetworkRank(self, n: int, roads) -> int:
4. degree, ans = {}, 0
5. **for** x **in** range(n):
6. degree[x] = 0
7. Edges = set()
8. **for** x, y **in** roads:
9. degree[x] += 1
10. degree[y] += 1
11. Edges.add((x, y))
12. Edges.add((y, x))
14. Nodes = [x **for** x **in** range(n)]
15. Nodes.sort(key=**lambda** x: -degree.get(x, 0))
16. second\_point = 0
17. **while** second\_point < n:
18. **if** degree[Nodes[second\_point]] == degree[Nodes[0]]:
19. second\_point += 1
20. **else**:
21. **break**
22. ans = 0
23. **if** second\_point == 1:
24. **for** x **in** Nodes[1:]:
25. ans = max(
26. ans, degree[x] + degree[Nodes[0]] -\
27. ((Nodes[0], x) **in** Edges)
28. )
29. **else**:
30. **if** second\_point \* (second\_point - 1) / 2 > len(roads):
31. **return** degree[Nodes[0]] + degree[Nodes[0]]
32. **else**:
33. **for** x **in** range(second\_point):
34. **for** y **in** range(x + 1, second\_point):
35. ans = max(
36. ans, degree[Nodes[x]] + degree[Nodes[y]] - \
37. ((Nodes[x], Nodes[y]) **in** Edges)
38. )
39. **return** ans

253. **[5122. Mean of Array After Removing Some Elements](https://leetcode-cn.com/problems/mean-of-array-after-removing-some-elements/)**

1. **class** Solution:
2. **def** trimMean(self, arr: List[int]) -> float:
3. arr.sort()
4. n = int(len(arr))
5. **return** sum(arr[n // 20: n - n // 20]) / (n \* 0.9)

254. **[5543. Largest Substring Between Two Equal Characters](https://leetcode-cn.com/problems/largest-substring-between-two-equal-characters/)**

1. **class** Solution:
2. **def** maxLengthBetweenEqualCharacters(self, s: str) -> int:
3. first, ans = {}, -1
4. **for** x **in** range(len(s)):
5. **if** s[x] **in** first:
6. ans = max(ans, x - first[s[x]] - 1)
7. **else**:
8. first[s[x]] = x
9. **return** ans

255. **[5544. Lexicographically Smallest String After Applying Operations](https://leetcode-cn.com/problems/lexicographically-smallest-string-after-applying-operations/)**

1. **class** Solution:
2. **def** findLexSmallestString(self, s: str, a: int, b: int) -> str:
3. **def** rotate(x, step):
4. n = len(x)
5. **return** x[n - step:] + x[: n - step]
6. Ans, n = s, len(s)
7. **for** x **in** range(10):
8. **for** y **in** range(10 **if** b & 1 **else** 1):
9. **for** step **in** range(n):
10. ans= ""
11. **for** t **in** range(len(s)):
12. **if** t & 1:
13. ans += "{}".format((int(s[t]) + x \* a) % 10)
14. **else**:
15. ans += "{}".format((int(s[t]) + y \* a) % 10)
16. ans = rotate(ans, (step \* b) % n)
17. **if** int(Ans) > int(ans):
18. Ans = ans
19. **return** Ans

256. **[LCP 02. 分式化简](https://leetcode-cn.com/problems/deep-dark-fraction/)**

1. **class** Solution:
2. **def** fraction(self, cont: List[int]) -> List[int]:
3. **import** math
4. cont = cont[::-1]
5. Frac = [cont[0], 1]
6. **for** x **in** cont[1:]:
7. Frac[0], Frac[1] = Frac[1], Frac[0]
8. Frac[0] += x \* Frac[1]
9. GCD = math.gcd(Frac[0], Frac[1])
10. **return** [Frac[0] // GCD, Frac[1] // GCD]

257. **[LCP 06. 拿硬币](https://leetcode-cn.com/problems/na-ying-bi/)**

1. **class** Solution:
2. **def** minCount(self, coins: List[int]) -> int:
3. **return** sum((x >> 1) + (1 & x) **for** x **in** coins)

258. **[LCP 07. 传递信息](https://leetcode-cn.com/problems/chuan-di-xin-xi/)**

1. **class** Solution:
2. **def** numWays(self, n: int, relation: List[List[int]], k: int) -> int:
3. **def** dfs(X, nowk, K, goal, successors):
4. **if** nowk == K:
5. **return** 1 **if** X == goal **else** 0
6. ans = 0
7. **for** x **in** successors.get(X, []):
8. ans += dfs(x, nowk + 1, K, goal, successors)
9. **return** ans
11. successors = {}
12. **for** x, y **in** relation:
13. **if** x **in** successors:
14. successors[x].append(y)
15. **else**:
16. successors[x] = [y]
17. **return** dfs(0, 0, k, n - 1, successors)

259. **[LCP 11. 期望个数统计](https://leetcode-cn.com/problems/qi-wang-ge-shu-tong-ji/)**

1. **class** Solution:
2. **def** expectNumber(self, scores: List[int]) -> int:
3. **return** len(set(scores))

260. **[LCP 18. 早餐组合](https://leetcode-cn.com/problems/2vYnGI/)**

1. **class** Solution:
2. **def** breakfastNumber(self, staple: List[int], drinks: List[int], x: int) -> int:
3. staple.sort()
4. drinks.sort()
5. ans = 0
6. now = len(drinks) - 1
7. **for** t **in** staple:
8. res = x - t
9. **while** now >= 0 **and** drinks[now] > res:
10. now -= 1
11. ans += (now + 1)
12. **return** ans % int(1e9 + 7)

261. **[LCP 22. 黑白方格画](https://leetcode-cn.com/problems/ccw6C7/)**

1. **class** Solution:
2. **def** paintingPlan(self, n: int, k: int) -> int:
3. **from** collections **import** Counter
4. **if** k == n \* n:
5. **return** 1
6. possible = {}
7. frac = [1]
8. **for** x **in** range(1, 7):
9. frac.append(frac[-1] \* x)
10. **for** x **in** range(0, n + 1):
11. **for** y **in** range(0, n + 1):
12. black = x \* n + (n - x) \* y
13. **if** black **in** possible:
14. possible[black] += (frac[n] / frac[x] / frac[n - x]) \* \
15. (frac[n] / frac[y] / frac[n - y])
16. **else**:
17. possible[black] = (frac[n] / frac[x] / frac[n - x]) \* \
18. (frac[n] / frac[y] / frac[n - y])
19. **return** int(possible.get(k, 0))

**262. 1154**

class Solution:

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

year = int(data[0:4])

month = int(data[5:7])

day = int(data[8:])

dic = [31,28,31,30,31,30,31,31,30,31,30,31]

if year%400==0 or year%4==0 and year%100!=0:

dic[1]=29

return sum(dic[:month-1])+day

**263. 1160**

class Solution:

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

chars\_cnt = collections.Counter(chars)

ans = 0

for word in words:

word\_cnt = collections.Counter(word)

for c in word\_cnt:

if chars\_cnt[c] < word\_cnt[c]:

break

else:

ans += len(word)

return ans

**264. 1170**

class Solution:

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

def func(s):

Hash = {}

for i in s:

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

return Hash[min(s)]

for k in range(len(queries)):

queries[k] = func(queries[k])

for l in range(len(words)):

words[l] = func(words[l])

Hash\_len = {1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0}

for key in Hash\_len.keys():

for m in words:

if m > key:

Hash\_len[key] += 1

result = []

for n in range(len(queries)):

result.append(Hash\_len[queries[n]])

return result

**265. 1175**

class Solution:

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

if n in [1, 2]:

return 1

comp = 1

prime = 1

ans = 1

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

for j in range(2, i):

if i % j == 0:

comp += 1

ans = ans \* comp % (10 \*\* 9 + 7)

break

else:

prime += 1

ans = ans \* prime % (10 \*\* 9 + 7)

return ans

**266. 1184**

class Solution:

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

calc\_distance = sum(distance[destination: start]) if start > destination else sum(distance[start: destination])

return min(calc\_distance, sum(distance)-calc\_distance)

**267. 1185**

import datetime

class Solution:

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

week\_day\_dict = {

0:"Sunday",

1:"Monday",

2:"Tuesday",

3:"Wednesday",

4:"Thursday",

5: "Friday",

6:"Saturday"

}

anyday=datetime.datetime(year,month,day).strftime("%w")

return week\_day\_dict[int(anyday)]

**268. 1189**

class Solution:

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

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

**269. 1200**

class Solution:

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

arr.sort()

min\_map = collections.defaultdict(list)

arry\_min\_val = float("inf")

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

min\_val = abs(arr[i+1]-arr[i])

arry\_min\_val = min(min\_val,arry\_min\_val)

min\_map[min\_val].append([arr[i],arr[i+1]])

return min\_map[arry\_min\_val]

**270. 1207**

class Solution:

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

dic={}

for i in arr:

if i in dic.keys():

dic[i] += 1

else:

dic[i] = 1

return len(dic.values())==len(set(dic.values()))

**271. 1217**

class Solution:

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

odd = sum(1 for i in chips if i % 2 == 1)

return min(odd,len(chips) - odd)

**272. 1221**

class Solution:

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

res = 0

a = 0

count\_L, count\_R = 0, 0

while a < len(s):

if s[a] == 'L':

count\_L += 1

else:

count\_R += 1

if count\_L == count\_R:

res += 1

count\_L = count\_R = 0

a += 1

return res

**273. 1232**

class Solution:

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

pt1 = coordinates[0]

pt2 = coordinates[1]

flag = 0

if len(set([i[0] for i in coordinates])) == 1 or len(set([i[1] for i in coordinates])) == 1: return True

elif pt2[0]-pt1[0] == 0: return False

else:

a = (pt2[1]-pt1[1])/(pt2[0]-pt1[0])

b = pt1[1]-pt1[0]\*a

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

if a\*coordinates[i][0] + b != coordinates[i][1]:

flag = 1

break

return flag == 0

**274. 1252**

class Solution:

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

res = 0

# 用 False 构造矩阵，因为默认0是偶数

arr = [[False] \* m for \_ in range(n)]

# 模拟矩阵 +1 操作，+1 意味着取反

for row, col in indices:

for j in range(m):

arr[row][j] = not arr[row][j]

for i in range(n):

arr[i][col] = not arr[i][col]

# 遍历矩阵，数True的个数

for row in arr:

for i in row:

if i:

res += 1

return res

**275. 1266**

class Solution(object):

def minTimeToVisitAllPoints(self, points):

"""

:type points: List[List[int]]

:rtype: int

"""

c=0

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

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

return c

**276. 1277**

class Solution:

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

dp,nums=matrix,0

for i in range(len(matrix)):

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

if matrix[i][j]==1:

if i>=1 and j>=1:

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

nums+=dp[i][j]

return nums

**277. 1281**

class Solution:

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

add, mul = 0, 1

while n > 0:

digit = n % 10

n //= 10

add += digit

mul \*= digit

return mul - add

**278. 1287**

class Solution:

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

n = len(arr)

cur, cnt = arr[0], 0

for i in range(n):

if arr[i] == cur:

cnt += 1

if cnt \* 4 > n:

return cur

else:

cur, cnt = arr[i], 1

return -1

**279. 1291**

class Solution:

def sequentialDigits(self, low: int, high: int) -> List[int]:

ans = list()

for i in range(1, 10):

num = i

for j in range(i + 1, 10):

num = num \* 10 + j

if low <= num <= high:

ans.append(num)

return sorted(ans)

**280. 1295**

class Solution:

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

return sum(1 for num in nums if len(str(num)) % 2 == 0)

**281. 1297**

class Solution:

def maxFreq(self, s: str, maxLetters: int, minSize: int, maxSize: int) -> int:

n = len(s)

occ = collections.defaultdict(int)

ans = 0

for i in range(n):

exist = set()

cur = ""

for j in range(i, min(n, i + maxSize)):

exist.add(s[j])

if len(exist) > maxLetters:

break

cur += s[j]

if j - i + 1 >= minSize:

occ[cur] += 1

ans = max(ans, occ[cur])

return ans

**282. 1299**

class Solution:

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

n = len(arr)

ans = [0] \* (n - 1) + [-1]

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

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

return ans

**283. 1-Two Sum**

class Solution:

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

waiting = {}

for index, num in enumerate(nums):

if target - num in waiting:

pair = [waiting[target-num], index]

return pair

else:

waiting[num] = index

**284. 3- Longest Substring Without Repeating Characters**

class Solution:

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

state = {}

left = 0

max\_len = 0

for right, ch in enumerate(s):

while state.get(ch, 0) != 0:

previous\_ch = s[left]

state[previous\_ch] = state.get(previous\_ch, 0) - 1

left += 1

# print(left, right, state)

state[ch] = state.get(ch, 0) + 1

max\_len = max(max\_len, right - left + 1)

return max\_len

**285. 5-Longest Palindromic Substring**

class Solution:

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

max\_len = 0

max\_str = ""

# case 1: odd length

for mid in range(len(s)):

half\_len = 0

while mid - half\_len >= 0 and mid + half\_len < len(s):

if s[mid - half\_len] == s[mid + half\_len]:

half\_len += 1

else:

break

# print(mid, half\_len)

half\_len -= 1

total\_len = 2 \* half\_len + 1

if total\_len > max\_len:

max\_str = s[mid-half\_len: mid+half\_len+1]

max\_len = total\_len

# print(max\_str, max\_len)

# case 2: even length

for mid\_left in range(len(s)-1):

mid\_right = mid\_left + 1

half\_len = 1

while mid\_left - half\_len + 1>= 0 and mid\_right + half\_len - 1 < len(s):

if s[mid\_left - half\_len + 1] == s[mid\_right + half\_len - 1]:

half\_len += 1

else:

break

# print(mid\_left, half\_len)

half\_len -= 1

total\_len = 2 \* half\_len

# print(total\_len)

if total\_len > max\_len:

max\_str = s[mid\_left - half\_len + 1: mid\_right + half\_len - 1 + 1]

max\_len = total\_len

return max\_str

**286. 6- ZigZag Conversion**

class Solution:

def convert(self, s: str, numRows: int) -> str:

if numRows == 1: # we do not need to care about direction here

return s

rows = [""] \* numRows

walk\_down = True

temp\_row = 0

for ch in s:

rows[temp\_row] += ch

if walk\_down:

if temp\_row == numRows-1: # last row

walk\_down = False # turn upwards

temp\_row -= 1

else:

temp\_row += 1

else:

if temp\_row == 0: # first row

walk\_down = True # turn downwards

temp\_row += 1

else:

temp\_row -= 1

return "".join(rows)

**287. 13. Roman to Integer**

class Solution:

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

symbol\_value = {

"M": 1000, "CM": 900,

"D": 500, "CD": 400,

"C": 100, "XC": 90,

"L": 50, "XL": 40,

"X": 10, "IX":9,

"V": 5, "IV": 4,

"I": 1

}

index = 0

result = 0

while index < len(s):

if index+1 < len(s) and s[index: index+2] in symbol\_value:

result += symbol\_value[s[index: index+2]]

index += 2

elif s[index] in symbol\_value:

result += symbol\_value[s[index]]

index += 1

else:

pass # output error

return result

**288. 17. Letter Combinations of a Phone Number**

class Solution:

def letterCombinations(self, digits: str) -> List[str]:

if not digits:

return []

digit\_letter = {

"2": "abc", "3": "def",

"4": "ghi", "5": "jkl", "6": "mno",

"7": "pqrs", "8": "tuv", "9": "wxyz"

}

result\_till\_now = [""]

for digit in digits:

new\_result = []

temp\_len = len(result\_till\_now)

for letter in digit\_letter[digit]:

for i in range(temp\_len):

new\_result.append(result\_till\_now[i] + letter)

result\_till\_now = new\_result

return result\_till\_now

**289. 20- Valid Parentheses**

class Solution:

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

left\_brace = ["(", "{", "["]

right\_brace = [")", "}", "]"]

stack = []

for brace in s:

if brace in left\_brace:

stack.append(brace)

else:

if not stack:

return False

elif left\_brace.index(stack[-1]) != right\_brace.index(brace):

return False

else:

stack.pop()

return len(stack) == 0

290. **36- Valid Sudoku**

class Solution:

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

self.board = board

for i in range(9):

if not self.valid\_row(i):

return False

if not self.valid\_column(i):

return False

if not self.valid\_block(i):

return False

return True

def valid\_row(self, row\_index):

row = self.board[row\_index]

counter = set()

for j in range(9):

if row[j] == ".":

continue

elif row[j] in counter:

return False

else:

counter.add(row[j])

return True

def valid\_column(self, col\_index):

counter = set()

for i in range(9):

if self.board[i][col\_index] == ".":

continue

elif self.board[i][col\_index] in counter:

return False

else:

counter.add(self.board[i][col\_index])

return True

def valid\_block(self, block\_index):

counter = set()

for block\_row in range(block\_index // 3 \* 3, (block\_index // 3 + 1) \* 3):

for block\_col in range(block\_index % 3 \* 3, (block\_index % 3 + 1) \* 3):

if self.board[block\_row][block\_col] == ".":

continue

elif self.board[block\_row][block\_col] in counter:

return False

else:

counter.add(self.board[block\_row][block\_col])

return True

**291. 46- Permutations**

class Solution:

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

if not nums:

return []

result\_till\_now = [[]]

for num in nums:

new\_result = []

for result in result\_till\_now:

result\_copy = result.copy()

for i in range(len(result)):

temp\_result = result\_copy[: i] + [num] + result\_copy[i:]

new\_result.append(temp\_result)

result\_copy = result.copy()

result\_copy.append(num)

new\_result.append(result\_copy)

result\_till\_now = new\_result

# print(result\_till\_now)

return result\_till\_now

**292. 48. Rotate Image**

class Solution:

def rotate(self, matrix: List[List[int]]) -> None:

"""

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

"""

n = len(matrix)

for i in range(n // 2):

for j in range((n+1) // 2):

num1 = matrix[i][j]

num2 = matrix[j][n-i-1]

num3 = matrix[n-i-1][n-j-1]

num4 = matrix[n-j-1][i]

# num1, num2, num3, num4 = num4, num1, num2, num3

matrix[i][j] = num4

matrix[j][n-i-1] = num1

matrix[n-i-1][n-j-1] = num2

matrix[n-j-1][i] = num3

**293. 49- Group Anagrams**

class Solution:

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

groups = {}

for temp\_str in strs:

temp\_key = [0] \* 26

for ch in temp\_str:

temp\_key[ord(ch) - ord('a')] += 1

group\_key = tuple(temp\_key)

if group\_key in groups:

groups[group\_key].append(temp\_str)

else:

groups[group\_key] = [temp\_str]

return list(groups.values())

**294. 53- Maximum Subarray**

You may simply test all subarrays. The complexity would be O(n2)

Or, a better solution of O(n):

class Solution:

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

left = 0

max\_sum = max(nums)

temp\_sum = 0

for right, num in enumerate(nums):

temp\_sum += num

if temp\_sum < 0:

left = right+1

temp\_sum = 0

else:

max\_sum = max(temp\_sum, max\_sum)

return max\_sum

**295. 54- Spiral Matrix**

class Solution:

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

if not matrix or not matrix[0]:

return []

min\_row = 0

min\_col = 0

max\_row = len(matrix) -1

max\_col = len(matrix[0]) - 1

numbers = (max\_row+1) \* (max\_col+1)

if max\_row == 0:

return matrix[0]

if max\_col == 0:

return [row[0] for row in matrix]

temp\_direction = "right"

temp\_row = 0

temp\_col = 0

result = []

for i in range(numbers):

# print(temp\_row, temp\_col, temp\_direction)

# print(min\_row, max\_row, min\_col, max\_col)

result.append(matrix[temp\_row][temp\_col])

if temp\_direction == "right":

if temp\_col != max\_col:

temp\_col += 1

else:

min\_row += 1

temp\_direction = "down"

temp\_row += 1

elif temp\_direction == "left":

if temp\_col != min\_col:

temp\_col -= 1

else:

temp\_direction = "up"

temp\_row -= 1

max\_row -= 1

elif temp\_direction == "down":

if temp\_row != max\_row:

temp\_row += 1

else:

temp\_direction = "left"

temp\_col -= 1

max\_col -= 1

else:

if temp\_row != min\_row:

temp\_row -= 1

else:

temp\_direction = "right"

temp\_col += 1

min\_col += 1

return result

**296. 55- Jump Game**

class Solution:

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

rightmost = 0

for index, num in enumerate(nums):

if rightmost >= len(nums) - 1:

return True

if rightmost < index: # can not reach index

return False

rightmost = max(rightmost, index+num)

**297. 56- Merge Intervals**

class Solution:

def merge(self, intervals: List[List[int]]) -> List[List[int]]:

result = []

intervals.sort()

for (start, end) in intervals:

if not result or result and result[-1][1] < start:

# no previous one, or no overlap with previous one

result.append([start, end])

else:

result[-1][1] = max(end, result[-1][1])

return result

**298. 59- Spiral Matrix II**

class Solution:

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

if n == 1:

return [[1]]

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

min\_row = 0

min\_col = 0

max\_row = n-1

max\_col = n-1

numbers = (max\_row+1) \* (max\_col+1)

if max\_row == 0:

return matrix[0]

if max\_col == 0:

return [row[0] for row in matrix]

temp\_direction = "right"

temp\_row = 0

temp\_col = 0

result = []

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

# print(temp\_row, temp\_col, temp\_direction)

# print(min\_row, max\_row, min\_col, max\_col)

matrix[temp\_row][temp\_col] = i

if temp\_direction == "right":

if temp\_col != max\_col:

temp\_col += 1

else:

min\_row += 1

temp\_direction = "down"

temp\_row += 1

elif temp\_direction == "left":

if temp\_col != min\_col:

temp\_col -= 1

else:

temp\_direction = "up"

temp\_row -= 1

max\_row -= 1

elif temp\_direction == "down":

if temp\_row != max\_row:

temp\_row += 1

else:

temp\_direction = "left"

temp\_col -= 1

max\_col -= 1

else:

if temp\_row != min\_row:

temp\_row -= 1

else:

temp\_direction = "right"

temp\_col += 1

min\_col += 1

return matrix

**299. 62- Unique Paths**

class Solution:

def uniquePaths(self, m: int, n: int) -> int:

self.cache = {}

return self.helper(m-1, n-1)

def helper(self, row, col):

if row == 0 or col == 0:

return 1

else: # row != 0 and col != 0

if (row, col) in self.cache:

return self.cache[(row, col)]

result = self.helper(row-1, col) + self.helper(row, col-1)

self.cache[(row, col)] = result

return result

**300. 71- Simplify Path**

class Solution:

def simplifyPath(self, path: str) -> str:

commands = path.split("/")

stack = []

for command in commands:

if not command or command == ".":

continue

elif command == "..":

if not stack:

continue

else:

stack.pop()

else:

stack.append(command)

return "/" + "/".join(stack)

**301. 75- Sort Colors**

class Solution:

def sortColors(self, nums: List[int]) -> None:

"""

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

"""

counters = [0, 0, 0]

for num in nums:

counters[num] += 1

for i in range(counters[0]):

nums[i] = 0

for j in range(counters[0], counters[0] + counters[1]):

nums[j] = 1

for k in range(counters[0] + counters[1], sum(counters)):

nums[k] = 2

**302. 89- Gray Code**

class Solution:

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

result = [0]

for i in range(n):

new\_result = []

for num in result[::-1]:

new\_result.append(num + (1 << i))

result.extend(new\_result)

# print(result)

return result