1.Counting Elements Given an integer array arr, count how many elements x there are, such that x + 1 is also in arr. If there are duplicates in arr, count them separately

def count\_elements(arr):

return sum(1 for x in arr if x + 1 in arr)

arr = [1, 2, 3]

print(count\_elements(arr))

2. You are given a string s containing lowercase English letters, and a matrix shift, where shift[i] = [directioni, amounti]:

Sol:-

class Solution:

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

total\_shift = 0

for sh in shift:

if sh[0] == 0:

total\_shift -= sh[1]

else:

total\_shift += sh[1]

total\_shift %= len(s)

return s[-total\_shift:] + s[:-total\_shift]

3. Leftmost Column with at Least a One A row-sorted binary matrix means that all elements are 0 or 1 and each row of the matrix is sorted in non-decreasing order.

Sol:-

class Solution:

def leftMostColumnWithOne(self, binaryMatrix: 'BinaryMatrix') -> int:

rows, cols = binaryMatrix.dimensions()

current\_row = 0

current\_col = cols - 1

leftmost\_col = -1

while current\_row < rows and current\_col >= 0:

if binaryMatrix.get(current\_row, current\_col) == 1:

leftmost\_col = current\_col

current\_col -= 1

else:

current\_row += 1

return leftmost\_col

4. You have a queue of integers, you need to retrieve the first unique integer in the queue. Implement the FirstUnique class:

from collections import OrderedDict

class FirstUnique:

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

self.queue = OrderedDict()

self.unique\_nums = {}

for num in nums:

self.add(num)

def showFirstUnique(self):

if self.queue:

return next(iter(self.queue.values()))

return -1

def add(self, value):

if value in self.unique\_nums:

if self.unique\_nums[value]:

self.queue.pop(value)

self.unique\_nums[value] = False

else:

self.queue[value] = value

self.unique\_nums[value] = True

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree.

Sol:-

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def isValidSequence(root, arr):

def check\_path(node, index):

if not node or node.val != arr[index]:

return False

if index == len(arr) - 1:

return not node.left and not node.right

return check\_path(node.left, index + 1) or check\_path(node.right, index + 1)

return check\_path(root, 0)

root = TreeNode(0)

root.left = TreeNode(1)

root.right = TreeNode(0)

root.left.left = TreeNode(0)

root.left.right = TreeNode(1)

root.right.left = None

root.right.right = None

root.left.left.left = None

root.left.left.right = None

root.left.right.left = TreeNode(1)

root.left.right.right = TreeNode(0)

arr = [0, 1, 0, 1]

print(isValidSequence(root, arr))

6. There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have

def kidsWithCandies(candies, extraCandies):

max\_candies = max(candies)

return [candy + extraCandies >= max\_candies for candy in candies

6. There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have.

def kidsWithCandies(candies, extraCandies):

max\_candies = max(candies)

return [candy + extraCandies >= max\_candies for candy in candies]

candies = [2, 3, 5, 1, 3]

extraCandies = 3

output = kidsWithCandies(candies, extraCandies)

print(output)

7. Max Difference You Can Get From Changing an Integer You are given an integer num. You will apply the following steps exactly two times:

def maxDiff(num):

s = str(num)

a = int(s.replace(max(s), '9'))

b = int(s.replace(min(s), '1' if s[0] != '1' else '0'))

return a - b

num = 555

print(maxDiff(num))

8. Check If a String Can Break Another String Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa.

def checkIfCanBreak(s1, s2):

return all(x >= y for x, y in zip(sorted(s1), sorted(s2))) or all(x <= y for x, y in zip(sorted(s1), sorted(s2)))

s1 = "abc"

s2 = "xya"

print(checkIfCanBreak(s1, s2))

9. Number of Ways to Wear Different Hats to Each Other There are n people and 40 types of hats labeled from 1 to 40. Given a 2D integer array hats, where hats[i] is a list of all hats preferred by the ith person. Return the number of ways that the n people wear different hats to each other. Since the answer may be too large, return it modulo 109 + 7.

Sol:-

def numberWays(hats):

MOD = 10\*\*9 + 7

n = len(hats)

dp = [0] \* (1 << n)

dp[0] = 1

hat\_to\_people = [[] for \_ in range(41)]

for i, h in enumerate(hats):

for j in h:

hat\_to\_people[j].append(i)

for hat in range(1, 41):

new\_dp = dp[:]

for state in range(1 << n):

for person in hat\_to\_people[hat]:

if state & (1 << person):

continue

new\_state = state | (1 << person)

new\_dp[new\_state] += dp[state]

new\_dp[new\_state] %= MOD

dp = new\_dp

return sum(dp) % MOD

10. You are given the array paths, where paths[i] = [cityAi, cityBi] means there exists a direct path going from cityAi to cityBi. Return the destination city, that is, the city without any path outgoing to another city.

Sol:- def destCity(paths):

start\_cities = set()

end\_cities = set()

for path in paths:

start\_cities.add(path[0])

end\_cities.add(path[1])

return (end\_cities - start\_cities).pop()

paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]

print(destCity(paths))