ASSIGNMENT:-4

1. 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

Sol:- def count\_elements(arr):

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

arr = [1, 2, 3]

print(count\_elements(arr))

1. 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]

1. 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

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

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

1. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree. We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a sequence in the given binary tree.

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)

1. 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

Sol:- def kidsWithCandies(candies, extraCandies):

max\_candies = max(candies)

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

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

Sol:-

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))

1. 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.

Sol:- 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(s2), sorted(s1)))

s1 = "abc"

s2 = "xya"

print(checkIfCanBreak(s1, s2))

1. 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.

Sol:- from functools import lru\_cache

class Solution:

def numberWays(self, hats: List[List[int]]) -> int:

MOD = 10\*\*9 + 7

n = len(hats)

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

for i, h in enumerate(hats):

for hat in h:

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

@lru\_cache(None)

def dp(used, mask):

if mask == (1 << n) - 1:

return 1

if used == 40:

return 0

res = dp(used + 1, mask)

for p in hat\_to\_people[used]:

if mask & (1 << p):

continue

res += dp(used + 1, mask | (1 << p))

res %= MOD

return res

return dp(0, 0)

1. A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

Sol:- def nextPermutation(nums):

i = len(nums) - 2

while i >= 0 and nums[i + 1] <= nums[i]:

i -= 1

if i >= 0:

j = len(nums) - 1

while j >= 0 and nums[j] <= nums[i]:

j -= 1

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

nums[i + 1:] = nums[i + 1:][::-1]