DAY- 4

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

Program:

```
def count_elements(arr):
  element_set = set(arr)
  count = 0
  for element in arr:
    if element + 1 in element_set:
      count += 1
  return count
arr = [1, 2, 3, 4, 5, 6]
print(count_elements(arr))
```

Output: 5

2. Perform String Shifts

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

Program:

```
def perform_string_shifts(s, shift):
  net_shift = 0
  for direction, amount in shift:
    if direction == 0:
      net_shift -= amount  # Left shift
```

```
        else:
            net_shift += amount  # Right shift

    net_shift = net_shift % len(s)
    if net_shift == 0:
        return s
    elif net_shift > 0:
        return s[-net_shift:] + s[:-net_shift]
    else:
        return s[-net_shift:] + s[:-net_shift]  # This handles negative shifts as well
s = "abcdefg"
shift = [[1, 1], [1, 1], [0, 2], [1, 3]]
print(perform_string_shifts(s, shift))
```

Output: "efgabcd"

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. Given a row-sorted binary matrix binaryMatrix, return the index (0-indexed) of the leftmost column with a 1 in it. If such an index does not exist, return -1. You can't access the Binary Matrix directly. You may only access the matrix using a BinaryMatrix interface:

Program:

```
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
```

Program:

```
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.

Program:

```
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.

Program:

```
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:

Program:

```
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.

Program:

```
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.

Program:

```
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.

Program:

```
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))
```