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.

Example:

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

Explanation: 1 and 2 are counted cause 2 and 3 are in arr.

Program:

```python
def count_elements(arr):

    element_set = set(arr)
    count = 0

    for x in arr:
        if x + 1 in element_set:
            count += 1

    return count

arr = [1, 2, 3]
print(f"Input: arr = {arr}")
print(f"Output: {count_elements(arr)}")

arr2 = [1, 1, 2, 2]
print(f"Input: arr = {arr2}")
print(f"Output: {count_elements(arr2)}")  # O

arr3 = [1, 3, 2, 3, 5, 0]
print(f"Input: arr = {arr3}")
print(f"Output: {count_elements(arr3)}")  # O

arr4 = [1, 1, 1, 1]
print(f"Input: arr = {arr4}")
print(f"Output: {count_elements(arr4)}")  # O
```

```
Output
Input: arr = [1, 2, 3]
Output: 2
Input: arr = [1, 1, 2, 2]
Output: 2
Input: arr = [1, 3, 2, 3, 5, 0]
Output: 3
Input: arr = [1, 1, 1, 1]
Output: 0

=== Code Execution Successful ===
```

2. Perform String Shifts

You are given a string s containing lowercase English letters, and a matrix shift, where shift[i] = [directioni, amounti]:
Example 1: Input: s = "abc", shift = [[0,1],[1,2]] Output: "cab"
Explanation: [0,1] means shift to left by 1. "abc" -> "bca" [1,2] means shift to right by 2. "bca" -> "cab"

Program:

```python
def perform_string_shifts(s, shift):
    net_shift = 0
    for direction, amount in shift:
        if direction == 0:
            net_shift -= amount
        else:
            net_shift += amount

    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]
s = "abc"
shift = [[0, 1], [1, 2]]
print(f"Input: s = \"{s}\", shift = {shift}")
print(f"Output: \"{perform_string_shifts(s, shift)}\"")  # Output: "cab

s2 = "abcdefg"
shift2 = [[1, 1], [1, 1], [0, 2], [1, 3]]
print(f"Input: s = \"{s2}\", shift = {shift2}")
print(f"Output: \"{perform_string_shifts(s2, shift2)}\"")  # Output: "e

s3 = "abcdefg"
shift3 = [[0, 1], [0, 1], [0, 1], [0, 7]]
print(f"Input: s = \"{s3}\", shift = {shift3}")
print(f"Output: \"{perform_string_shifts(s3, shift3)}\"")  # Output: "a

s4 = "abcdefg"
shift4 = [[1, 1], [0, 2], [1, 2], [0, 1]]
print(f"Input: s = \"{s4}\", shift = {shift4}")
print(f"Output: \"{perform_string_shifts(s4, shift4)}\"")  # Output: "g
```

Output

```
Input: s = "abc", shift = [[0, 1], [1, 2]]
Output: "cab"
Input: s = "abcdefg", shift = [[1, 1], [1, 1], [0, 2], [1, 3]]
Output: "efgabcd"
Input: s = "abcdefg", shift = [[0, 1], [0, 1], [0, 1], [0, 7]]
Output: "defgabc"
Input: s = "abcdefg", shift = [[1, 1], [0, 2], [1, 2], [0, 1]]
Output: "abcdefg"

=== Code Execution Successful ===
```

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: • BinaryMatrix.get(row, col) returns the element of the matrix at index (row, col) (0-indexed).
• BinaryMatrix.dimensions() returns the dimensions of the matrix as a list of 2 elements [rows, cols], which means the matrix is rows x cols.
Example 1:
Input: mat = [[0,0],[1,1]] Output:

```python
class BinaryMatrix:
    def __init__(self, matrix):
        self.matrix = matrix
        self.rows = len(matrix)
        self.cols = len(matrix[0])
    def get(self, row, col):
        return self.matrix[row][col]
    def dimensions(self):
        return [self.rows, self.cols]
def leftmost_column_with_one(binaryMatrix):
    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
mat = [[0, 0], [1, 1]]
binaryMatrix = BinaryMatrix(mat)
print(f"Output: {leftmost_column_with_one(binaryMatrix)}")  # Output: 0
mat2 = [[0, 0, 0, 1], [0, 0, 1, 1], [0, 1, 1, 1]]
binaryMatrix2 = BinaryMatrix(mat2)
print(f"Output: {leftmost_column_with_one(binaryMatrix2)}")  # Output: 1
mat3 = [[0, 0], [0, 0]]
binaryMatrix3 = BinaryMatrix(mat3)
print(f"Output: {leftmost_column_with_one(binaryMatrix3)}")  # Output: -
mat4 = [[0, 0, 0, 0, 1]]
binaryMatrix4 = BinaryMatrix(mat4)
print(f"Output: {leftmost_column_with_one(binaryMatrix4)}")  # Output: 4
```

Output

```
Output: 0
Output: 1
Output: -1
Output: 4

=== Code Execution Successful ===
```

4. First Unique Number

You have a queue of integers, you need to retrieve the first unique integer in the queue.

Implement the FirstUnique class:

- **FirstUnique(int[] nums) Initializes the object with the numbers in the queue.**
- **int showFirstUnique() returns the value of the first unique integer of the queue.**
- **void add(int value) insert value to the queue.**

Example 1:

Input:["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique","a dd","showFirstUnique"] [[[2,3,5]],[],[5],[],[2],[],[3],[]]

Output: [null,2,null,2,null,3,null,-1]

Explanation: FirstUnique firstUnique = new FirstUnique([2,3,5]); firstUnique.showFirstUnique(); // return 2 firstUnique.add(5); // the queue is now [2,3,5,5] firstUnique.showFirstUnique(); // return 2 firstUnique.add(2); // the queue is now [2,3,5,5,2] firstUnique.showFirstUnique(); // return 3 firstUnique.add(3); // the queue is now [2,3,5,5,2,3] firstUnique.showFirstUnique(); // return -1

Program:

```python
from collections import deque, defaultdict
class FirstUnique:
    def __init__(self, nums):
        self.queue = deque()
        self.counts = defaultdict(int)
        for num in nums:
            self.add(num)

    def showFirstUnique(self):
        while self.queue and self.counts[self.queue[0]] > 1:
            self.queue.popleft()
        if self.queue:
            return self.queue[0]
        else:
            return -1

    def add(self, value):
        self.counts[value] += 1
        if self.counts[value] == 1:
            self.queue.append(value)

firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique())  # Output: 2
firstUnique.add(5)
print(firstUnique.showFirstUnique())  # Output: 2
firstUnique.add(2)
print(firstUnique.showFirstUnique())  # Output: 3
firstUnique.add(3)
print(firstUnique.showFirstUnique())  # Output: -1
```

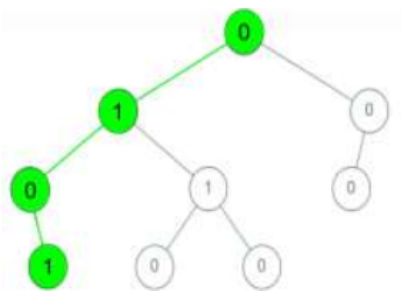
Output

```
2
2
3
-1

=== Code Execution Successful ===
```

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree Given a binary tree where each path going from the root to any leaf form a valid sequence, check if a given string is a valid sequence in such binary tree. We get the given string from the concatenation of an array of integers arr and the concatenation of all values of the nodes along a path results in a sequence in the given binary tree.
Example 1: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,0,1] Output: true Explanation: The path 0 -> 1 -> 0 -> 1 is a valid sequence (green color in the figure). Other valid sequences are: 0 -> 1 -> 1 -> 0 0 -> 0 -> 0



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 dfs(node, index):
        if not node:
            return False
        if node.val != arr[index]:
            return False
        if index == len(arr) - 1:
            return node.left is None and node.right is None
        return dfs(node.left, index + 1) or dfs(node.right, index + 1)
    return dfs(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 = TreeNode(0)
root.left.left.right = TreeNode(1)
root.left.right.left = TreeNode(0)
root.left.right.right = TreeNode(0)

arr = [0, 1, 0, 1]
print(isValidSequence(root, arr))
```

```
Output
True

=== Code Execution Successful
```

6. Kids With the Greatest Number of Candies

There are n kids with candies. You are given an integer array candies, where each candies[i] represents the number of candies the ith kid has, and an integer extraCandies, denoting the number of extra candies that you have. Return a boolean array result of length n, where result[i] is true if, after giving the ith kid all the extraCandies, they will have the greatest number of candies among all the kids, or false otherwise.Note that multiple kids can have the greatest number of candies.

Example 1: Input: candies = [2,3,5,1,3], extraCandies = 3 Output: [true,true,true,false,true] Explanation: If you give all extraCandies to:

Kid 1, they will have 2 + 3 = 5 candies, which is the greatest among the kids.
Kid 2, they will have 3 + 3 = 6 candies, which is the greatest among the kids.
Kid 3, they will have 5 + 3 = 8 candies, which is the greatest among the kids.

Program:

```python
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)  # Find the maximum ca
    result = []

    for candy in candies:
        if candy + extraCandies >= max_candies:
            result.append(True)
        else:
            result.append(False)

    return result

# Example usage
candies = [2, 3, 5, 1, 3]
extraCandies = 3
print(kidsWithCandies(candies, extraCandies))  # Outp
```

```
Output
[True, True, True, False, True]

=== Code Execution Successful ===
```

7. Max Difference You Can Get From Changing an Integer

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

• Pick a digit x (0 <= x <= 9).

• Pick another digit y (0 <= y <= 9). The digit y can be equal to x.

• Replace all the occurrences of x in the decimal representation of num by y.

• The new integer cannot have any leading zeros, also the new integer cannot be 0. Let a and b be the results of applying the operations to num the first and second times, respectively. Return the max difference between a and z.
Example 1: Input: num = 555 Output: 888

Explanation: The first time pick x = 5 and y = 9 and store the new integer in a. The second time pick x = 5 and y = 1 and store the new integer in b. We have now a = 999 and b = 111 and max difference = 888.

Program:

```python
def maxDifference(num):

    num_str = str(num)

    max_num_str = num_str
    for digit in num_str:
        if digit != '9':
            max_num_str = num_str.replace(digit, '9')
            break
    max_num = int(max_num_str)

    min_num_str = num_str
    if num_str[0] != '1':
        min_num_str = num_str.replace(num_str[0], '1')
    else:
        for digit in num_str:
            if digit != '0' and digit != '1':
                min_num_str = num_str.replace(digit, '
                break
    min_num = int(min_num_str)

    return max_num - min_num

num = 555
print(maxDifference(num))  # Output: 888
```

```
Output

888

=== Code Execution Successful
```

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. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.

Example 1: Input: s1 = "abc", s2 = "xya" Output: true

Explanation: "ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc".

Program:

```python
def checkIfCanBreak(s1, s2):
    # Sort both strings
    sorted_s1 = sorted(s1)
    sorted_s2 = sorted(s2)

    # Check if sorted_s1 can break sorted_s2
    can_s1_break_s2 = all(c1 >= c2 for c1, c2 in zip(sorted_s1, sorted_s2))

    # Check if sorted_s2 can break sorted_s1
    can_s2_break_s1 = all(c2 >= c1 for c1, c2 in zip(sorted_s1, sorted_s2))

    return can_s1_break_s2 or can_s2_break_s1

# Example usage
s1 = "abc"
s2 = "xya"
print(checkIfCanBreak(s1, s2))  # Output: true
```

Output

```
True

=== Code Execution Successful
```

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.
Example 1: Input: hats = [[3,4],[4,5],[5]] Output: 1
Explanation: There is only one way to choose hats given the conditions. First person choose hat 3, Second person choose hat 4 and last one hat 5.

Program:

```python
def numberWays(hats):
    MOD = 10**9 + 7
    n = len(hats)

    hat_to_people = {}
    for person, hat_list in enumerate(hats):
        for hat in hat_list:
            if hat not in hat_to_people:
                hat_to_people[hat] = []
            hat_to_people[hat].append(person)

    dp = [0] * (1 << n)
    dp[0] = 1  # One way to assign zero hats

    for hat in range(1, 41):
        if hat in hat_to_people:
```

Output

```
1

=== Code Execution Successful ===
```

```
            new_dp = dp[:]
            for state in range(1 << n):
                for person in hat_to_people[hat]:

                    if state & (1 << person) == 0:

                        new_state = state | (1 << person)
                        new_dp[new_state] = (new_dp[new_state] + dp[state]) % MOD
            dp = new_dp

    return dp[(1 << n) - 1]

hats = [[3, 4], [4, 5], [5]]
print(numberWays(hats))  # Output: 1
```

10. Next Permutation

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.
Given an array of integers nums, find the next permutation of nums. The replacement must be in place and use only constant extra memory.
Example 1: Input: nums = [1,2,3] Output: [1,3,2]

Program:

```
def nextPermutation(nums):
    n = len(nums)
    if n <= 1:
        return

    i = n - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1

    if i >= 0:

        j = n - 1
        while nums[j] <= nums[i]:
            j -= 1

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

    nums[i + 1:] = reversed(nums[i + 1:])

nums = [1, 2, 3]
nextPermutation(nums)
print(nums)
```

Output

```
[1, 3, 2]

=== Code Execution Successful
```