ASSIGNMENT -3

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

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

Counting Elements

VSC > Assignment 2 > Counting Elements > ...

```python
def countElements(arr):
    count = 0
    seen = set(arr)
    for num in arr:
        if num + 1 in seen:
            count += 1
    return count

arr = [1, 2, 3]
result = countElements(arr)
print(result)  # Output: 2
```

Output

```
2

=== 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]: ● directioni can be 0 (for left shift) or 1 (for right shift). ● amounti is the amount by which string s is to be shifted. ● A left shift by 1 means remove the first character of s and append it to the end. ● Similarly, a right shift by 1 means remove the last character of s and add it to the beginning. Return the final string after all operations.

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"

Perform string shifts

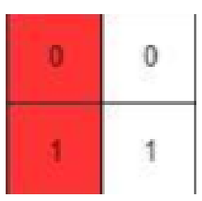
VSC > Assignment 2 > Perform string shifts

```python
def stringShifts(s, shift):
    total_shift = 0
    for direction, amount in shift:
        if direction == 0:
            total_shift -= amount
        else:
            total_shift += amount
    total_shift %= len(s)
    if total_shift > 0:
        return s[-total_shift:] + s[:-total_shift]
    else:
        return s[abs(total_shift):] + s[:abs(total_shift)]

s = "abc"
shift = [[0, 1], [1, 2]]
result = stringShifts(s, shift)
print(result)  # Output: "cab"
```

Output

```
cab

=== 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. Submissions making more than 1000 calls to BinaryMatrix.get will be judged Wrong Answer. Also, any solutions that attempt to circumvent the judge will result in disqualification. For custom testing purposes, the input will be the entire binary matrix mat. You will not have access to the binary matrix directly.



Example 1: Input: mat = [[0,0],[1,1]] Output: 0

Leftmost column

VSC > Leftmost column > ...

```python
def leftmostColumnWithOne(binaryMatrix):
    rows, cols = len(binaryMatrix), len(binaryMatrix[0])
    min_leftmost = cols
    for row in range(rows):
        leftmost_one = cols
        left, right = 0, cols - 1
        while left <= right:
            mid = (left + right) // 2
            if binaryMatrix[row][mid] == 1:
                leftmost_one = mid
                right = mid - 1
            else:
                left = mid + 1
        min_leftmost = min(min_leftmost, leftmost_one)
    return min_leftmost if min_leftmost < cols else -1

matrix = [[0, 0], [1, 1]]
result = leftmostColumnWithOne(matrix)
print(result)
```

Output

```
0

=== 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, and returns -1 if there is no such integer. ● void add(int value) insert value to the queue.

Example 1: Input: ["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique","add","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

First couting nums ●

VSC > Assignment 2 > First couting nums > ...

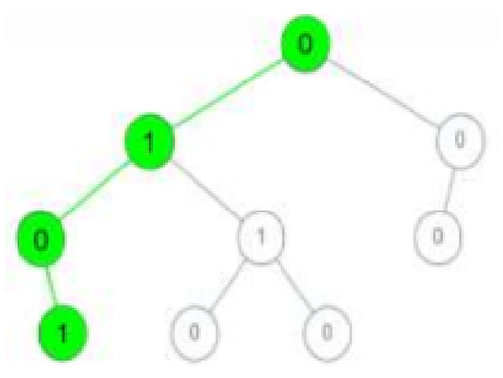
```python
from collections import deque
class FirstUnique:
    def __init__(self, nums):
        self.queue = deque(nums)
        self.count_map = {}
        for num in nums:
            self.count_map[num] = self.count_map.get(num, 0) + 1
    def showFirstUnique(self):
        while self.queue and self.count_map[self.queue[0]] > 1:
            self.queue.popleft()
        if self.queue:
            return self.queue[0]
        else:
            return -1
    def add(self, value):
        self.count_map[value] = self.count_map.get(value, 0) + 1
        self.queue.append(value)

firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique())
firstUnique.add(5)
print(firstUnique.showFirstUnique())
firstUnique.add(2)
print(firstUnique.showFirstUnique())
firstUnique.add(3)
print(firstUnique.showFirstUnique())
```

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

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

valid string from roots to node ●

VSC > Assignment 2 > valid string from roots to node > ...

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

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

root = TreeNode(0)
root.left = TreeNode(1)
root.right = TreeNode(0)
root.left.left = TreeNode(0)
root.left.right = TreeNode(1)
root.right.right = TreeNode(0)
arr = [0,1,0,1]
result = isValidSequence(root, arr)
print(result)
```

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. - Kid 4, they will have 1 + 3 = 4 candies, which is not the greatest among the kids. - Kid 5, they will have 3 + 3 = 6 candies, which is the greatest among the kids.

kids with great num of candy's

VSC > Assignment 2 > kids with great num of candy's > ...

```python
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)
    result = [candy + extraCandies >= max_candies for candy in candies]
    return result

candies = [2, 3, 5, 1, 3]
extraCandies = 3
result = kidsWithCandies(candies, extraCandies)
print(result)
```

Output

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

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

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: ● 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 b.

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

Max difference

VSC > Assignment 2 > Max difference > ...

```python
def maxDiff(num):
    if num < 0:
        return maxDiff(-num)
    elif num == 0:
        return 88
    num_str = str(num)
    n = len(num_str)
    max_diff = 0
    for i in range(n):
        if num_str[i] == '0' and i == 0:
            continue
        for digit in range(10):
            if digit == int(num_str[i]):
                continue
            new_num_str = num_str[:i] + str(digit) + num_str[i + 1:]
            new_num = int(new_num_str) * (1 if i == 0 else 10**(n - i - 1))
            diff = abs(num - new_num)
            max_diff = max(max_diff, diff)
    return max_diff

num = 555
result = maxDiff(num)
print(result)  # Output: 888
```

```
Output

5395

=== 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".

if string can break another string.py ●

VSC > Assignment 2 > if string can break another string.py > ...

```python
def canBreak(s1, s2):
    n = len(s1)
    count1, count2 = 0, 0
    for i in range(n):
        if s1[i] > s2[i]:
            count1 += 1
        elif s1[i] < s2[i]:
            count2 += 1
    return count1 > 0 and count2 == 0

s1 = "abc"
s2 = "xya"
result = canBreak(s1, s2)
print(result)

s1 = "abc"
s2 = "yxb"
result = canBreak(s1, s2)
print(result)
```

Output

```
False
False

=== 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.

num of ways to wear different hats.py ●

VSC > Assignment 2 > num of ways to wear different hats.py > ...

```python
from functools import lru_cache

MOD = 10 ** 9 + 7

@lru_cache(None)
def dp(hats, person, assigned):
    n = len(hats)
    if person == n:
        return 1

    ways = 0
    for hat in hats[person]:
        if hat not in assigned:
            ways += dp(hats, person + 1, assigned + (hat,))

    return ways % MOD

def waysToWearHats(hats):
    return dp(tuple(tuple(h) for h in hats), 0, ())

hats = [[3, 4], [4, 5], [5]]
print(waysToWearHats(hats))
```

Output

```
1

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

10. Next Permutation A permutation of an array of integers is an arrangement of its members into a sequence or linear order. ● For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]. The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container. If such arrangement is

not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order). ● For example, the next permutation of arr = [1,2,3] is [1,3,2]. ● Similarly, the next permutation of arr = [2,3,1] is [3,1,2]. ● While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement. 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]

Next permutation.py

VSC > Assignment 2 > Next permutation.py > ...

```python
def nextPermutation(nums):
    n = len(nums)
    i = n - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1
    if i == -1:
        nums.reverse()
        return
    j = i + 1
    while j < n and nums[j] > nums[i]:
        j += 1
    nums[i], nums[j - 1] = nums[j - 1], nums[i]
    nums[i + 1:] = nums[i + 1:][::-1]

nums = [1, 2, 3]
nextPermutation(nums)
print(nums)  # Output: [1, 3, 2]

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

Output

```
[1, 3, 2]
[1, 2, 3]

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