ASSIGNMENT-3

Name: G. venkata Praveen

Reg.no:192373023

Dept:cse(D.s)

1. 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.

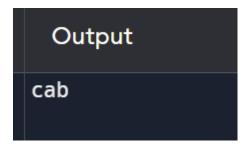
```
1 - def count_elements(arr):
    count = 0
 3
    num_set = set(arr)
 4
 5 for num in arr:
 6  if num + 1 in num_set:
7
    count += 1
 8
 9
    return count
10
11
    arr = [1, 2, 3]
   result = count_elements(arr)
12
13
   print(result) # Output: 2
```

Output:



- 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.

```
1 - def string_shift(s, shift):
    total_shift = 0
2
3 for direction, amount in shift:
4 if direction == 0:
   total_shift -= amount
5
6 else:
7 total_shift += amount
    total_shift %= len(s)
    return s[-total_shift:] + s[:-total_shift]
9
10
   s = "abc"
11
12
   shift = [[0, 1], [1, 2]]
   output = string_shift(s, shift)
13
   print(output) # Output: "cab"
14
```



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.

```
1 class Solution:
    def leftMostColumnWithOne(self, binaryMatrix: 'BinaryMatrix') ->
        int:
    rows, cols = binaryMatrix.dimensions()
3
    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
10
11 else:
    current_row += 1
12
13
    return leftmost_col
```

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.

```
1 class FirstUnique:
2 def __init__(self, nums):
3
    self.queue = []
    self.unique_dict = OrderedDict()
 5 for num in nums:
    self.add(num)
7 def showFirstUnique(self):
8 - if self.unique_dict:
    return next(iter(self.unique_dict.values()))
10
    return -1
11 def add(self, value):
12 - if value in self.unique_dict:
    self.unique_dict.pop(value)
13
14 - elif value not in self.queue:
    self.unique_dict[value] = value
16
    self.queue.append(value)
17
   firstUnique = FirstUnique([2, 3, 5])
18
   print(firstUnique.showFirstUnique()) # Output: 2
19
   firstUnique.add(5)
20
   print(firstUnique.showFirstUnique()) # Output: 2
21
22 firstUnique.add(2)
   print(firstUnique.showFirstUnique()) # Output: 3
```

```
Output

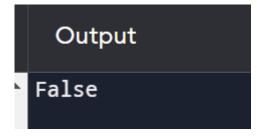
2
2
3
-1
```

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.

```
1 - class TreeNode:
 2 def __init__(self, val=0, left=None, right=None):
 3 self.val = val
 4 self.left = left
   self.right = right
 6 def is_valid_sequence(root, arr):
 7 def check_path(node, index):
 8 if not node or index == len(arr) or node.val != arr[index]:
    return False
10 - if not node.left and not node.right and index == len(arr) - 1:
11
    return True
12
    return check_path(node.left, index + 1) or check_path(node
         .right, index + 1)
13
    return check_path(root, 0)
14
15
   root = TreeNode(0)
16 root.left = TreeNode(1)
17 root.right = TreeNode(0)root.left.left = TreeNode(0)
18 root.left.right = TreeNode(1)
19 root.right.left = None
20 root.right.right = None
21
   arr = [0, 1, 0, 1]
22 print(is_valid_sequence(root, arr)) # Output: True
```

Output:



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

```
1 - def distribute_candies(candies, extra_candies):
2 try:
    max_candies = max(candies)
3
    result = [candy + extra_candies >= max_candies for candy in
        candies]
5
   return result
6 - except Exception as e:
7 print(f"An error occurred: {e}")
8
   return None
9 # Example
10 candies = [2, 3, 5, 1, 3]
   extra_candies = 3
12 output = distribute_candies(candies, extra_candies)
13 print(output)
```

Output:

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

- 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).
- ullet 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

```
1 def max_diff(num):
2 def replace_digit(n, x, y):
   return int(str(n).replace(str(x), str(y)))
   max_diff_result = 0
5 for x in range(10):
6 for y in range(10):
   a = replace_digit(num, x, 9)
   b = replace_digit(a, y, 1)
9 	imes if a != 0 and b != 0 and a - b > max_diff_result:
0
   max_diff_result = a - b
1
   return max_diff_result
   num = 555
  output = max_diff(num)
4
5 print(output) # Output: 888
```



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.

```
1 def check_permutation_break(s1, s2):
2 if len(s1) != len(s2):
    raise ValueError("Both strings must be of the same length.")
3
4
 5
    s1_sorted = sorted(s1)
6
    s2_sorted = sorted(s2)
7
8
    if all(s1_char >= s2_char for s1_char, s2_char in zip(s1_sorted,
         s2_sorted)) or
9 all(s2_char >= s1_char for s1_char, s2_char in zip(s1_sorted,
       s2_sorted)):
10
    return True
11 else:
12 return False
13 # Test the function with the provided example
14 s1 = "abc"
15 s2 = "xya"
16 result = check_permutation_break(s1, s2)
17 print(result) # Output: True
```



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.

```
def num_ways_to_choose_hats(hats):
     MOD = 10**9 + 7
 2
    n = len(hats)
 3
    all hats = 1 << 40
 5
    dp = [0] * all_hats
 6
    dp[0] = 1
 7 for i in range(1, n + 1):
 8 for hat in hats[i - 1]:
 9 for j in range(all_hats - 1, -1, -1):
10 · if j & (1 << hat):
    dp[j] += dp[j \wedge (1 << hat)]
11
12 dp[j] %= MOD
13 return dp[-1]
14 # Example
15 hats = [[3, 4], [4, 5], [5]]
16 - try:
17 result = num_ways_to_choose_hats(hats)
    print("Number of ways to choose hats:", result)
19 - except Exception as e:
    print("An error occurred:", e)
20
```

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).
- \bullet For example, the next permutation of arr = [1,2,3] is [1,3,2].
- \bullet 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.

```
1 def next_permutation(nums):
 2 # Find the first decreasing element from the right
    i = len(nums) - 2
 4 while i \ge 0 and nums[i] \ge nums[i + 1]:
    i -= 1
 6
 7 - if i >= 0:
 8 # Find the next greater element to swap with
    j = len(nums) - 1
9
10 - while nums[j] <= nums[i]:</pre>
11
12
    nums[i], nums[j] = nums[j], nums[i]
13
14
15  nums[i + 1:] = nums[i + 1:][::-1]
16
17
   nums = [1, 2, 3]
18 print("Input:", nums)
19 next_permutation(nums)
20 print("Output:", nums)
```

Output

Input: [1, 2, 3]

Output: [1, 3, 2]