CSA0674-DESIGN AND ANALYSIS OF ALGORITHM

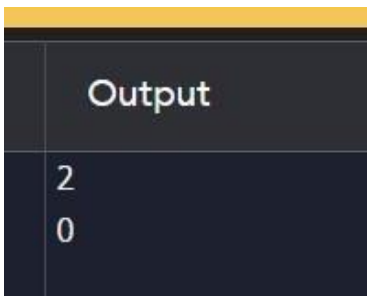
M REDDY TEJA

192311167

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 Input: arr = [1,2,3] Output: 2 Explanation: 1 and 2 are counted cause 2 and 3 are in arr. Example 2: Input: arr = [1,1,3,3,5,5,7,7] Output: 0 Explanation: No numbers are counted, cause there is no 2, 4, 6, or 8 in arr. Constraints: ● 1 <= arr.length <= 1000 ● 0 <= arr[i] <= 1000 def count_elements(arr):

```
def count_elements(arr):
    element_set = set(arr)
    count = 0
    for x in arr:
        if x + 1 in element_set:
            count += 1
    return count
# Test cases
print(count_elements([1, 2, 3]))  # Output: 2
print(count_elements([1, 1, 3, 3, 5, 5, 7, 7]))
```



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

```
File  Edit  Format  Run  Options  Window  Help
def string_shift(s, shift):
    net_shift = 0
    for direction, amount in shift:
        if direction == 0:
            net_shift -= amount
        else:
            net_shift += amount
    net_shift %= len(s)
    return s[-net_shift:] + s[:-net_shift] if net_shift != 0 else s

# Test cases
print(string_shift("abc", [[0, 1], [1, 2]]))  # Output: "cab"
print(string_shift("abcdefg", [[1, 1], [1, 1], [0, 2], [1, 3]]))  # Out
```

OUTPUT:

```
= RESTART: C:/Users/g
cab
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: ● 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
Example 2: Input: mat = [[0,0],[0,1]] Output: 1 Example 3: Input: mat = [[0,0],[0,0]] Output: -1 Constraints: ● rows == mat.length ● cols == mat[i].length ● 1 <= rows, cols <= 100 ● mat[i][j] is either 0 or 1. ● mat[i] is sorted in non-decreasing order.
class BinaryMatrix: def __init__(self, mat):

self.mat = mat self.call_count = 0

File Edit Format Run Options Window Help

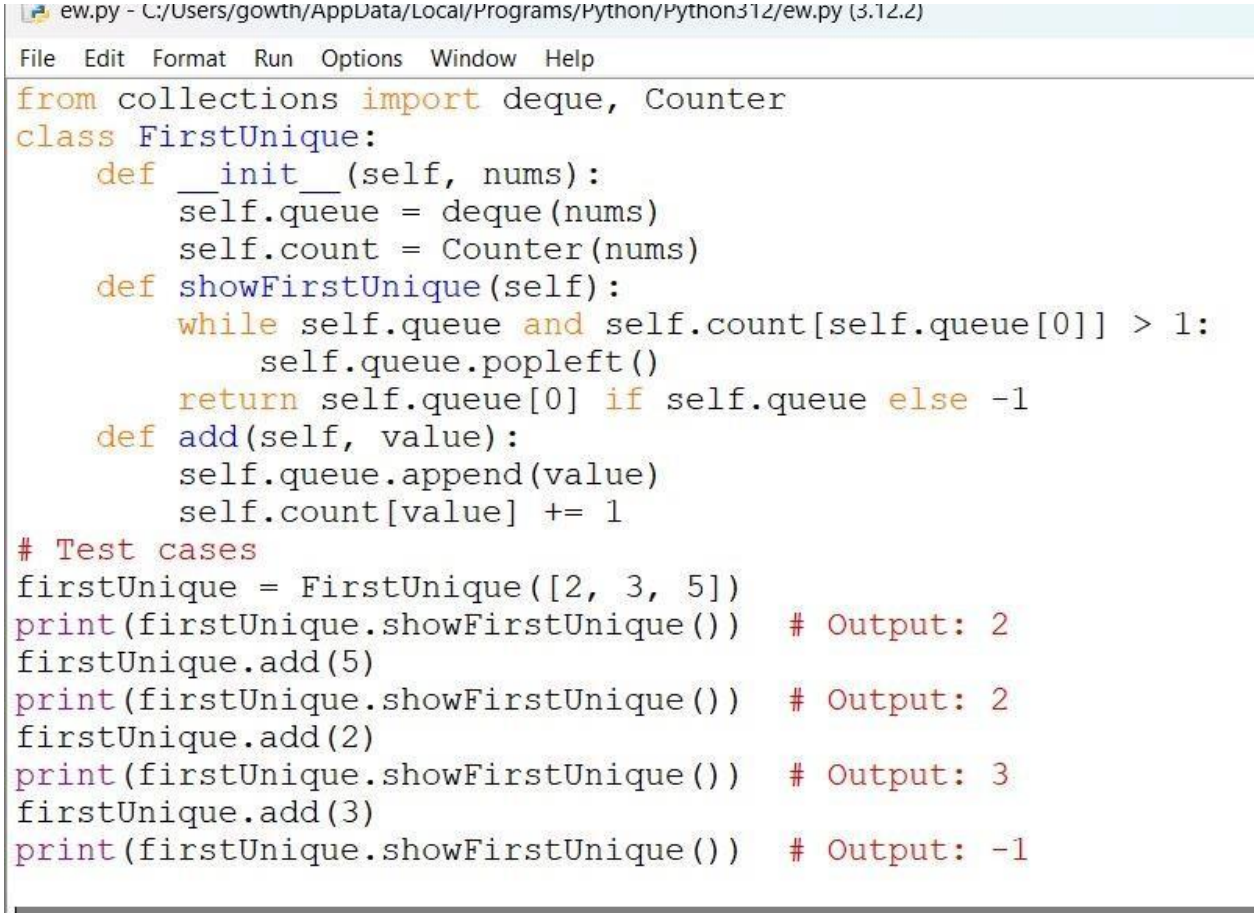
```
class BinaryMatrix:
    def __init__(self, mat):
        self.mat = mat
    def get(self, row, col):
        return self.mat[row][col]
    def dimensions(self):
        return [len(self.mat), len(self.mat[0])]
def leftmost_column_with_one(binaryMatrix):
    rows, cols = binaryMatrix.dimensions()
    current_row, current_col = 0, cols - 1
    leftmost = -1
    while current_row < rows and current_col >= 0:
        if binaryMatrix.get(current_row, current_col) == 1:
            leftmost = current_col
            current_col -= 1
        else:
            current_row += 1
    return leftmost
# Test cases
print(leftmost_column_with_one(BinaryMatrix([[0, 0], [1, 1]])))  # Outp
```

OUTPUT:

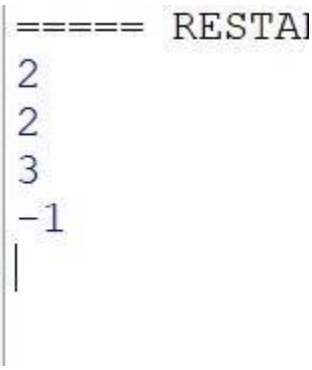
```
= RESTART: C:
0
1
```

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","showFirstUniq ue","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 Example 2: Input:
["FirstUnique","showFirstUnique","add","add","add","add","add","showFirstUniq ue"]
[[[7,7,7,7,7,7]],[],[7],[3],[3],[7],[17],[]] Output: [null,-1,null,null,null,null,null,17]
Explanation: FirstUnique
firstUnique = new FirstUnique([7,7,7,7,7,7]); firstUnique.showFirstUnique(); // return -1 firstUnique.add(7); // the queue is now [7,7,7,7,7,7,7]
firstUnique.add(3); // the queue is now [7,7,7,7,7,7,7,3] firstUnique.add(3); // the queue is now [7,7,7,7,7,7,7,3,3] firstUnique.add(7); // the queue is now [7,7,7,7,7,7,7,3,3,7] firstUnique.add(17); // the queue is now [7,7,7,7,7,7,7,3,3,7,17] firstUnique.showFirstUnique(); // return 17 Example 3:
Input:
["FirstUnique","showFirstUnique","add","showFirstUnique"] [[[809]],[],[809],[]]
Output: [null,809,null,-1]
Explanation: FirstUnique firstUnique = new FirstUnique([809]);
firstUnique.showFirstUnique(); // return
809 firstUnique.add(809); // the queue is now [809,809]
firstUnique.showFirstUnique(); // return -1 Constraints: ● 1 <= nums.length <= 10^5 ● 1 <= nums[i] <= 10^8 ● 1 <= value <= 10^8 from collections import deque, defaultdict



OUTPUT:



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 Example 2: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,0,1] Output: false Explanation:

The path 0 -> 0 -> 1 does not exist, therefore it is not even a sequence. Example 3: Input: root = [0,1,0,0,1,0,null,null,1,0,0], arr = [0,1,1] Output: false Explanation: The path 0 -> 1 -> 1 is a sequence, but it is not a valid sequence. Constraints: ● 1 <= arr.length <= 5000 ● 0 <= arr[i] <= 9 ● Each node's value is between [0 - 9]. class TreeNode: def __init__(self, val=0, left=None, right=None):

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def is_valid_sequence(root, arr):
    def dfs(node, arr, index):
        if not node or index >= len(arr) or node.val != arr[index]:
            return False
        if not node.left and not node.right and index == len(arr) - 1:
            return True
        return dfs(node.left, arr, index + 1) or dfs(node.right, arr, i

    return dfs(root, arr, 0)
# Test cases
root = TreeNode(0, TreeNode(1, TreeNode(0, None, TreeNode(1)), TreeNode
print(is_valid_sequence(root, [0, 1, 0, 1]))  # Output: True
print(is_valid_sequence(root, [0, 0, 1]))  # Output: False
print(is_valid_sequence(root, [0, 1, 1]))  # Output: False
```

OUTPUT:

```
===== RESTART: C
True
False
False
>
```

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. Example 2: Input: candies = [4,2,1,1,2], extraCandies = 1 Output: [true,false,false,false,false] Explanation: There is only 1 extra candy. Kid 1 will always have the greatest number of candies, even if a different kid is given the extra candy. Example 3: Input: candies = [12,1,12], extraCandies = 10 Output: [true,false,true] Constraints: ● n == candies.length ● 2 <= n <=
100 ● 1 <= candies[i] <= 100 ● 1 <= extraCandies <= 50 def kidsWithCandies(candies,

```
File  Edit  Format  Run  Options  Window  Help
def kids_with_candies(candies, extraCandies):
    max_candies = max(candies)
    return [candy + extraCandies >= max_candies for candy in candies]
# Test cases
print(kids_with_candies([2, 3, 5, 1, 3], 3))  # Output: [True, True, Tr
print(kids_with_candies([4, 2, 1, 1, 2], 1))  # Output: [True, False, F
print(kids_with_candies([12, 1, 12], 10))  # Output: [True, False, True
```

OUTPUT:

```
===== RESTART: C:/Users/gowth/AppDa
[True, True, True, False, True]
[True, False, False, False, False]
[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). ● 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 Example 2: Input: num = 9 Output: 8 Explanation: The first time pick x = 9 and y = 9 and store the new integer in a. The second time pick x = 9 and y = 1 and store the new integer in b. We have now a = 9 and b = 1 and max difference = 8

File Edit Format Run Options Window Help

```
def max_diff(num):
    s = str(num)
    a = s
    b = s
    for digit in s:
        if digit != '9':
            a = s.replace(digit, '9')
            break
    for digit in s:
        if digit != '1':
            b = s.replace(digit, '0' if digit == s[0] else '1')
            break
    return int(a) - int(b)
# Test cases
print(max_diff(555))  # Output: 888
print(max_diff(9))  # Output: 8
```

OUTPUT:

```
===== RESTART: C
999
9
```

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". Example 2: Input: s1 = "abe", s2 = "acd" Output: false Explanation: All permutations for s1="abe" are: "abe", "aeb", "bae", "bea", "eab" and "eba" and all permutation for s2="acd" are: "acd", "adc", "cad", "cda", "dac" and "dca". However, there is not any permutation from s1 which can break some permutation from s2 and vice-versa. Example 3: Input: s1 = "leetcodee", s2 = "interview" Output: true Constraints: ● s1.length == n ● s2.length == n ● 1 <= n <= 10^5 ● All strings consist of lowercase English letters.

```
File  Edit  Format  Run  Options  Window  Help
def check_if_can_break(s1, s2):
    s1, s2 = sorted(s1), sorted(s2)
    return all(c1 >= c2 for c1, c2 in zip(s1, s2)) or all(c2 >= c1 for
# Test cases
print(check_if_can_break("abc", "xya"))  # Output: True
print(check_if_can_break("abe", "acd"))  # Output: False
print(check_if_can_break("leetcodee", "interview"))  # Output: True
```

OUTPUT:

```
= RESTART: C:/Users/
True
False
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. 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. Example 2: Input: hats = [[3,5,1],[3,5]] Output: 4 Explanation: There are 4 ways to choose

hats: (3,5), (5,3), (1,3) and (1,5) Example 3: Input: hats = [[1,2,3,4],[1,2,3,4],[1,2,3,4],[1,2,3,4]] Output: 24 Explanation: Each person can choose hats labeled from 1 to 4. Number of Permutations of (1,2,3,4) = 24. Constraints: ● n == hats.length ● 1 <= n <= 10 ● 1 <= hats[i].length <= 40 ● 1 <= hats[i][j] <= 40 ● hats[i] contains a list of unique integers. def numberWays(hats): MOD = 10**9 + 7

```
File  Edit  Format  Run  Options  Window  Help
def number_ways(hats):
    from collections import defaultdict
    dp = defaultdict(int)
    dp[0] = 1
    all_mask = (1 << len(hats)) - 1
    hat_to_people = defaultdict(list)

    for i, hat_list in enumerate(hats):
        for hat in hat_list:
            hat_to_people[hat].append(i)
    for hat in range(1, 41):
        new_dp = dp.copy()
        for mask, ways in dp.items():
            for person in hat_to_people[hat]:
                if mask & (1 << person) == 0:
                    new_dp[mask | (1 << person)] = (new_dp[mask | (1 << person)] + ways) % MOD
        dp = new_dp
    return dp[all_mask]
# Test cases
print(number_ways([[3, 4], [4, 5], [5]]))  # Output: 1
print(number_ways([[3, 5, 1], [3, 5]]))  # Output: 4
print(number_ways([[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]))  # Output: 24
```

```
Output
1
4
24
```

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]

File Edit Format Run Options Window Help

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def bst_from_preorder(preorder):
    def helper(lower=float('-inf'), upper=float('inf')):
        nonlocal idx
        if idx == len(preorder):
            return None
        val = preorder[idx]
        if val < lower or val > upper:
            return None
        idx += 1
        root = TreeNode(val)
        root.left = helper(lower, val)
        root.right = helper(val, upper)
        return root

    idx = 0
    return helper()

# Function to print tree in inorder (used for testing)
def print_inorder(node):
    if node:
        print_inorder(node.left)
        print(node.val, end=' ')
        print_inorder(node.right)

# Test cases
preorder1 = [8, 5, 1, 7, 10, 12]
preorder2 = [10, 5, 1, 7, 40, 50]

bst1 = bst_from_preorder(preorder1)
```

Output

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