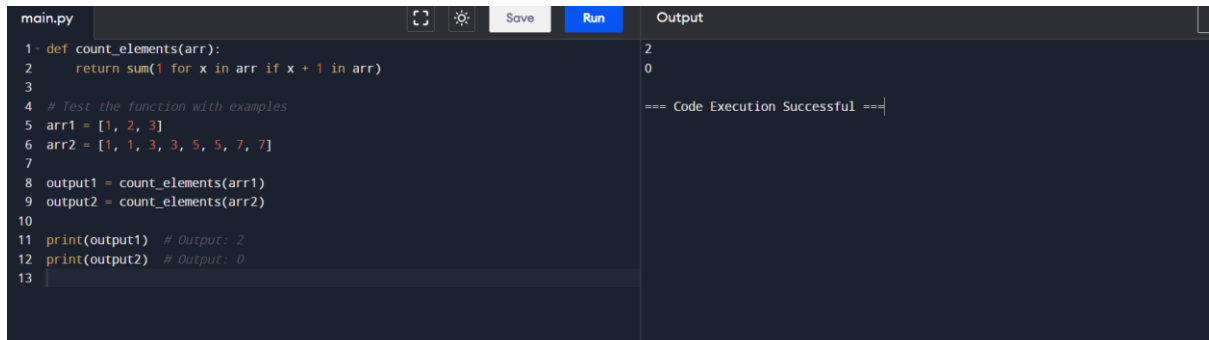


**1.Counting Elements** Given an integer array arr, count how many elements x there are, such that  $x + 1$  is also in arr. If there are duplicates in arr, count them separately.

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

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



```
main.py  [ ] [ ] Save Run Output
1 def count_elements(arr):
2     return sum(1 for x in arr if x + 1 in arr)
3
4 # Test the function with examples
5 arr1 = [1, 2, 3]
6 arr2 = [1, 1, 3, 3, 5, 5, 7, 7]
7
8 output1 = count_elements(arr1)
9 output2 = count_elements(arr2)
10
11 print(output1) # Output: 2
12 print(output2) # Output: 0
13
```

2  
0  
=== Code Execution Successful ===

**2.Perform String Shifts** You are given a string s containing lowercase English letters, and a matrix shift, where  $\text{shift}[i] = [\text{direction}_i, \text{amount}_i]$ :  
•  $\text{direction}_i$  can be 0 (for left shift) or 1 (for right shift).  
•  $\text{amount}_i$  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"

```

main.py  [ ] [ ] Save Run Output
1 def stringShift(s, shift):
2     total_shift = 0
3     for sh in shift:
4         if sh[0] == 0:
5             total_shift -= sh[1]
6         else:
7             total_shift += sh[1]
8
9     total_shift %= len(s)
10
11    return s[-total_shift:] + s[:-total_shift]
12
13    # Example 1
14    s1 = "abc"
15    shift1 = [[0,1],[1,2]]
16    print(stringShift(s1, shift1))
17
18    # Example 2
19    s2 = "abcdefg"
20    shift2 = [[1,1],[1,1],[0,2],[1,3]]
21    print(stringShift(s2, shift2))
22

```

cab  
efgabcd  
=== 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.

0	0
1	1

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

```

main.py  [ ] [ ] Save Run Output
1 def generate_row_sorted_binary_matrix(rows, cols):
2     import random
3     matrix = [[random.randint(0, 1) for _ in range(cols)] for _ in range(rows)]
4     for row in matrix:
5         row.sort()
6     return matrix
7
8    # Example Usage
9    rows = 2
10    cols = 2
11    binary_matrix = generate_row_sorted_binary_matrix(rows, cols)
12    print(binary_matrix)
13

```

[[0, 0], [0, 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

```
1  from collections import OrderedDict, deque
2
3  3 usages
4  class FirstUnique:
5      def __init__(self, nums):
6          self.queue = deque(nums)
7          self.count = OrderedDict()
8          for num in nums:
9              self.count[num] = self.count.get(num, 0) + 1
10
11      12 usages
12      def showFirstUnique(self):
13          for num in self.queue:
14              if self.count[num] == 1:
15                  return num
16          return -1
```

```

16         def add(self, value):
17             self.queue.append(value)
18             self.count[value] = self.count.get(value, 0) + 1
19
20     # Example 1
21     firstUnique = FirstUnique([2, 3, 5])
22     print(firstUnique.showFirstUnique()) # Output: 2
23     firstUnique.add(5)
24     print(firstUnique.showFirstUnique()) # Output: 2
25     firstUnique.add(2)
26     print(firstUnique.showFirstUnique()) # Output: 3
27     firstUnique.add(3)
28     print(firstUnique.showFirstUnique()) # Output: -1
29
30     # Example 2
31     firstUnique = FirstUnique([7, 7, 7, 7, 7, 7])
32     print(firstUnique.showFirstUnique()) # Output: -1

```

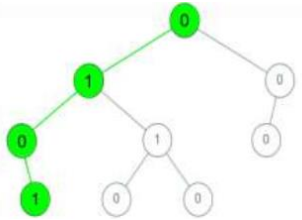
C:\Users\vinot

2  
2  
3  
-1  
-1  
-1  
3  
-1  
-1

-1  
3  
-1  
-1  
17  
809  
-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.



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

```
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, idx):
        if not node or idx == len(arr) or node.val != arr[idx]:
            return False

        if not node.left and not node.right and idx == len(arr) - 1:
            return True

        return dfs(node.left, idx + 1) or dfs(node.right, idx + 1)

    return dfs(root, idx=0)

# Example usage
root = TreeNode(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.right.right = TreeNode(0)

arr1 = [0, 1, 0, 1]
arr2 = [0, 0, 1]
arr3 = [0, 1, 1]

print(isValidSequence(root, arr1)) # Output: True
print(isValidSequence(root, arr2)) # Output: False
print(isValidSequence(root, arr3)) # Output: False

```

```

Run assign-3 Question x
"C:\Users\jacin\PycharmProjects\OAA-Design analysis of algorithm\.venv\Scripts\python.exe" "C:\Users\jacin\AppData\Roaming\JetBrains\PyCharmCE2023.3\src\
False
False
True
Process finished with exit code 0

```

**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  $i$ th 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  $i$ th 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.

```
main.py  [Icons] Save Run Output
1 def kidsWithCandies(candies, extraCandies):
2
3     max_candies = max(candies)
4
5     result = [candy + extraCandies >= max_candies for candy in candies]
6
7     return result
8
9 candies = [2, 3, 5, 1, 3]
10 extraCandies = 3
11 print(kidsWithCandies(candies, extraCandies))
12
13
```

```
[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 ( $0 \leq x \leq 9$ ). ● Pick another digit y ( $0 \leq y \leq 9$ ). 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

main.py	Save	Run	Output
<pre> 1 def max_diff(num): 2     num_str = str(num) 3     max_num = int('9' + num_str[1:].replace('9','8')) 4     min_num = int(('1' if num_str[0] != '1' else '0') + num_str[1:].replace       (num_str[0], '1')) 5     return max_num - min_num 6 print(max_diff(9)) </pre>			8  === 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  $x[i] \geq y[i]$  (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".

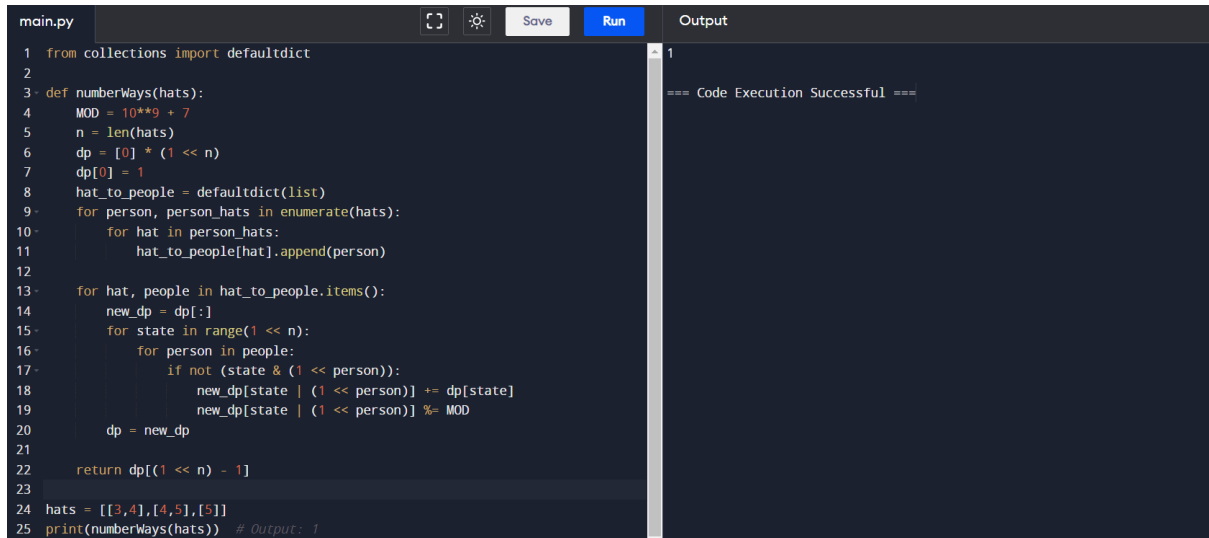
main.py	Save	Run	Output
<pre> 1 def checkIfCanBreak(s1, s2): 2     return all(ord(x) &lt;= ord(y) for x, y in zip(sorted(s1), sorted(s2))) 3 4 s1 = "abc" 5 s2 = "xya" 6 print(checkIfCanBreak(s1, s2)) </pre>			True  === Code Execution Successful ===

**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  $10^9 + 7$ .

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.



The screenshot shows a code editor with a file named 'main.py'. The code is a Python function 'numberWays(hats)' that calculates the number of ways to assign hats to people. It uses a dynamic programming approach with a dictionary 'hat\_to\_people' and a DP array 'dp'. The code is as follows:

```
1 from collections import defaultdict
2
3 def numberWays(hats):
4     MOD = 10**9 + 7
5     n = len(hats)
6     dp = [0] * (1 << n)
7     dp[0] = 1
8     hat_to_people = defaultdict(list)
9     for person, person_hats in enumerate(hats):
10         for hat in person_hats:
11             hat_to_people[hat].append(person)
12
13     for hat, people in hat_to_people.items():
14         new_dp = dp[:]
15         for state in range(1 << n):
16             for person in people:
17                 if not (state & (1 << person)):
18                     new_dp[state | (1 << person)] += dp[state]
19                     new_dp[state | (1 << person)] %= MOD
20         dp = new_dp
21
22     return dp[(1 << n) - 1]
23
24 hats = [[3,4],[4,5],[5]]
25 print(numberWays(hats)) # Output: 1
```

The output panel on the right shows '1' and '=== 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]

main.py	Save	Run	Output
<pre>1 def nextPermutation(nums): 2 3     i = len(nums) - 2 4     while i &gt;= 0 and nums[i + 1] &lt;= nums[i]: 5         i -= 1 6 7 8     if i &gt;= 0: 9         j = len(nums) - 1 10        while j &gt;= 0 and nums[j] &lt;= nums[i]: 11            j -= 1 12 13        nums[i], nums[j] = nums[j], nums[i] 14 15        nums[i + 1:] = nums[i + 1:][::-1] 16 17 # Example usage: 18 nums = [1, 2, 3] 19 nextPermutation(nums) 20 print(nums) 21</pre>			<pre>[1, 3, 2]  === Code Execution Successful ===</pre>