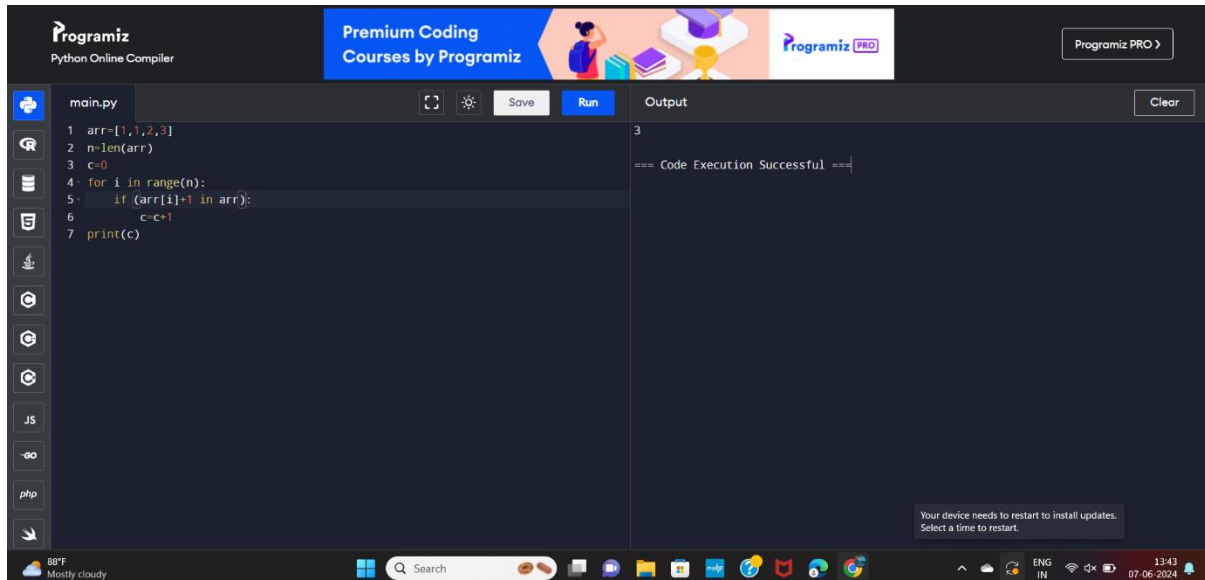


1. Counting Elements Given an integer array `arr`, count how many elements `x` there are, such that `x + 1` is also in `arr`. If there are duplicates in `arr`, count them separately. Example Input: `arr = [1,2,3]` Output: 2 Explanation: 1 and 2 are counted cause 2 and 3 are in `arr`. Example 2: Input: `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: •  $1 \leq \text{arr.length} \leq 1000$  •  $0 \leq \text{arr}[i] \leq 1000$



The screenshot shows the Programiz Python Online Compiler interface. The code editor on the left contains the following Python code:

```

1 arr=[1,1,2,3]
2 n=len(arr)
3 c=0
4 for i in range(n):
5     if (arr[i]+1 in arr):
6         c=c+1
7 print(c)

```

The output panel on the right shows the result of the code execution:

```

3

=== Code Execution Successful ===

```

The bottom of the image shows a Windows taskbar with the date and time 07-06-2024 13:43.

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" Example 2: Input: `s = "abcdefg"`, `shift = [[1,1],[1,1],[0,2],[1,3]]` Output: "efgabcd" Explanation: [1,1] means shift to right by 1. "abcdefg" -> "gabcdef" [1,1] means shift to right by 1. "gabcdef" -> "fgabcde" [0,2] means shift to left by 2. "fgabcde" -> "abcdefg" [1,3] means shift to right by 3. "abcdefg" -> "efgabcd"

The screenshot shows the Programiz Python Online Compiler interface. The code editor on the left contains a Python script named `main.py` with the following content:

```
1 from typing import List
2 class Solution:
3     def stringShift(self, s: str, shift: List[List[int]]) -> str:
4         x = sum((b if a else -b) for a, b in shift)
5         x %= len(s)
6         return s[-x:] + s[:-x]
```

The right-hand side of the interface shows the output area with the message: `=== Code Execution Successful ===`. The top navigation bar includes the Programiz logo, a 'Premium Coding Courses by Programiz' banner, and a 'Programiz PRO' button. The bottom status bar displays weather information (91°F, Mostly cloudy), a search bar, and system icons.

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]]`

The screenshot shows the Programiz Python Online Compiler interface with a solution for the 'Leftmost Column with at Least a One' problem. The code editor on the left contains the following Python code:

```
1 class BinaryMatrix:
2     def __init__(self, matrix):
3         self.matrix = matrix
4     def get(self, row, col):
5         return self.matrix[row][col]
6     def dimensions(self):
7         return len(self.matrix), len(self.matrix[0])
8 def leftMostColumnWithOne(binaryMatrix: 'BinaryMatrix') -> int:
9     rows, cols = binaryMatrix.dimensions()
10    current_row = 0
11    current_col = cols - 1
12    leftmost_col = -1
13    while current_row < rows and current_col >= 0:
14        if binaryMatrix.get(current_row, current_col) == 1:
15            leftmost_col = current_col
16            current_col -= 1
17        else:
18            current_row += 1
19    return leftmost_col
20 matrix = [[0,0],[1,1]]
21 binary_matrix = BinaryMatrix(matrix)
22 result = leftMostColumnWithOne(binary_matrix)
23 print(result)
24
```

The right-hand side of the interface shows the output area with the message: `0` and `=== Code Execution Successful ===`. The top navigation bar includes the Programiz logo, a 'Premium Coding Courses by Programiz' banner, and a 'Programiz PRO' button. The bottom status bar displays weather information (Humid Now), a search bar, and system icons.

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 deque, Counter
2
3 class unique:
4     def __init__(self, nums):
5         self.queue = deque(nums)
6         self.count = Counter(nums)
7
8     def showunique(self):
9         while self.queue and self.count[self.queue[0]] > 1:
10             self.queue.popleft()
11         return self.queue[0] if self.queue else -1
12
13     def add(self, value):
14         self.queue.append(value)
15         self.count[value] += 1
16
17 firstUnique = unique([2, 3, 5])
18 print(firstUnique.showunique())
19 firstUnique.add(5)
20 print(firstUnique.showunique())
21 firstUnique.add(2)
22 print(firstUnique.showunique())
23 firstUnique.add(3)
24 print(firstUnique.showunique())
  
```

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

```

1 class TreeNode:
2     def __init__(self, val=0, left=None, right=None):
3         self.val = val
4         self.left = left
5         self.right = right
6
7 def isValidSequence(root: TreeNode, arr: [int]) -> bool:
8     def dfs(node, idx):
9         if node is None:
10             return False
11         if idx >= len(arr):
12             return False
13         if node.val != arr[idx]:
14             return False
15         if node.left is None and node.right is None:
16             if idx == len(arr) - 1:
17                 return True
18             else:
19                 return False
20         if idx == len(arr) - 1:
21             return False
22         if dfs(node.left, idx + 1):
23             return True
24         if dfs(node.right, idx + 1):
25             return True
  
```

True

=== Code Execution Successful ===

```

24     return True
25     return False
26     return dfs(root, 0)
27 def buildTree(values):
28     if not values:
29         return None
30     root = TreeNode(values[0])
31     queue = [root]
32     i = 1
33     while queue and i < len(values):
34         node = queue.pop(0)
35         if values[i] is not None:
36             node.left = TreeNode(values[i])
37             queue.append(node.left)
38         i += 1
39         if i < len(values) and values[i] is not None:
40             node.right = TreeNode(values[i])
41             queue.append(node.right)
42         i += 1
43     return root
44 root = buildTree([0, 1, 0, 0, 1, 0, None, None, 1, 0, 0])
45 arr = [0, 1, 0, 1]
46 print(isValidSequence(root, arr))
47

```

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  $i$ 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]`

```

24     return True
25     return False
26     return dfs(root, 0)
27 def buildTree(values):
28     if not values:
29         return None
30     root = TreeNode(values[0])
31     queue = [root]
32     i = 1
33     while queue and i < len(values):
34         node = queue.pop(0)
35         if values[i] is not None:
36             node.left = TreeNode(values[i])
37             queue.append(node.left)
38         i += 1
39         if i < len(values) and values[i] is not None:
40             node.right = TreeNode(values[i])
41             queue.append(node.right)
42         i += 1
43     return root
44 root = buildTree([0, 1, 0, 0, 1, 0, None, None, 1, 0, 0])
45 arr = [0, 1, 0, 1]
46 print(isValidSequence(root, arr))
47

```

Output: 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
1 num_str = str(num)
2
3
4 def replace_digit(n_str, x, y):
5     return n_str.replace(x, y)
6
7 max_result = num
8 min_result = num
9 for digit in num_str:
10    if digit != '9':
11        max_str = replace_digit(num_str, digit, '9')
12        max_result = max(max_result, int(max_str))
13        break
14    if num_str[0] != '1':
15        min_str = replace_digit(num_str, num_str[0], '1')
16        min_result = min(min_result, int(min_str))
17    else:
18        for digit in num_str[1:]:
19            if digit != '0' and digit != num_str[0]:
20                min_str = replace_digit(num_str, digit, '0')
21                min_result = min(min_result, int(min_str))
22                break
23 return max_result - min_result
24 num = 555
25 print(maxDiff(num))

```

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  $x[i] \geq y[i]$  (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". 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.

```

main.py
1 def check(s1, s2):
2     s1_sorted = sorted(s1)
3     s2_sorted = sorted(s2)
4     can_s1_break_s2 = True
5     can_s2_break_s1 = True
6
7     for i in range(len(s1_sorted)):
8         if s1_sorted[i] < s2_sorted[i]:
9             can_s1_break_s2 = False
10        if s2_sorted[i] < s1_sorted[i]:
11            can_s2_break_s1 = False
12    return can_s1_break_s2 or can_s2_break_s1
13 s1 = "abc"
14 s2 = "xya"
15 print(check(s1, s2))
16

```

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  $i$ th 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.

```

1 def numberWays(hats):
2     MOD = 10**9 + 7
3     n = len(hats)
4     hat_to_people = [[] for _ in range(41)]
5     for person in range(n):
6         for hat in hats[person]:
7             hat_to_people[hat].append(person)
8     dp = [0] * (1 << n)
9     dp[0] = 1
10    for hat in range(1, 41):
11        for mask in range((1 << n) - 1, -1, -1):
12            for person in hat_to_people[hat]:
13                if mask & (1 << person) == 0:
14                    dp[mask | (1 << person)] += dp[mask]
15                    dp[mask | (1 << person)] %= MOD
16    return dp[(1 << n) - 1]
17    hats = [[3, 4], [4, 5], [5]]
18    print(numberWays(hats))

```

Output: 1

=== Code Execution Successful ===

10. Destination City You are given the array `paths`, where `paths[i] = [cityAi, cityBi]` means there exists a direct path going from `cityAi` to `cityBi`. Return the destination city, that is, the city without any path outgoing to another city. It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city. Example 1: Input: `paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]` Output: "Sao Paulo" Explanation: Starting at "London" city you will reach "Sao Paulo" city which is the destination city. Your trip consist of: "London" -> "New York" -> "Lima" -> "Sao Paulo".

```

1 def destCity(paths):
2     outgoing = set()
3     for path in paths:
4         outgoing.add(path[0])
5     for path in paths:
6         if path[1] not in outgoing:
7             return path[1]
8     paths = [["London", "New York"], ["New York", "Lima"], ["Lima", "Sao Paulo"]]
9     print(destCity(paths))
10

```

Output: Sao Paulo

=== Code Execution Successful ===