DAY-4-PROGRAMS

1.Counting Elements

PROGRAM:

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

2.Perform String Shifts

PROGRAM:

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

3. Leftmost Column with at Least a One

PROGRAM:

```
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

print(leftmost_column_with_one(BinaryMatrix([[0, 0], [1, 1]])))  # Output: 0
print(leftmost_column_with_one(BinaryMatrix([[0, 0], [0, 1]])))  # Output: 1
print(leftmost_column_with_one(BinaryMatrix([[0, 0], [0, 0]])))  # Output: -1
```

4. First Unique Number

PROGRAM:

```
from collections import deque, Counter
class FirstUnique:
    def __init__(self, nums):
        self.queue = deque(nums)
        self.count = Counter(nums)
    def showFirstUnique(self):
        while self.queue and self.count[self.queue[0]] > 1:
            self.queue.popleft()
        return self.queue[0] if self.queue else -1
    def add(self, value):
```

```
        self.queue.append(value)
        self.count[value] += 1


firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique())  # Output: 2
firstUnique.add(5)
print(firstUnique.showFirstUnique())  # Output: 2
firstUnique.add(2)
print(firstUnique.showFirstUnique())  # Output: 3
firstUnique.add(3)
print(firstUnique.showFirstUnique())  # Output: -1
```

5.Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree

PROGRAM:

```
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, index + 1)


    return dfs(root, arr, 0)


root = TreeNode(0, TreeNode(1, TreeNode(0, None, TreeNode(1)), TreeNode(1, TreeNode(0))),
TreeNode(0, TreeNode(0)))
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
```

6.Kids With the Greatest Number of Candies

PROGRAM:

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


print(kids_with_candies([2, 3, 5, 1, 3], 3))  # Output: [True, True, True, False, True]
print(kids_with_candies([4, 2, 1, 1, 2], 1))  # Output: [True, False, False, False, False]
print(kids_with_candies([12, 1, 12], 10))  # Output: [True, False, True]
```

7.Max Difference You Can Get From Changing an Integer

PROGRAM:

```
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
```

8.Check If a String Can Break Another String

PROGRAM:

```
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 c1, c2 in zip(s1, s2))
# 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
```

9.Number of Ways to Wear Different Hats to Each Other

PROGRAM :

```
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
```

10.Construct Binary Search Tree from Preorder Traversal

PROGRAM:

```
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)
bst2 = bst_from_preorder(preorder2)

print("Inorder traversal of BST from preorder1:")
print_inorder(bst1)  # Output: 1 5 7 8 10 12

print("\nInorder traversal of BST from preorder2:")
print_inorder(bst2)  # Output: 1 5 7 10 40 50
```