ASSIGNMENT- 03

DATE: 08-06-2024

1.Counting elements

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

2. Perform string shifts

```
def string_shifts(s, shift):
    total_shift = 0
    for direction, amount in shift:
        if direction == 0:
            total_shift -= amount
        else:
            total_shift += amount
    n = len(s)
    total_shift %= n
    return s[-total_shift:] + s[:-total_shift]
print(string_shifts("abc", [[0, 1], [1, 2]]))
```

3. Leftmost Column with at Least a One

```
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

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])]
binaryMatrix = BinaryMatrix([[0, 0], [1, 1]])
print(leftmost_column_with_one(binaryMatrix))
```

4. First unique number

```
from collections import deque
class FirstUnique:
  def _init_(self, nums):
      self.queue = deque()
      self.counts = {}
      for num in nums:
          self.add(num)

   def showFirstUnique(self):
      while self.queue and self.counts[self.queue[0]] > 1:
          self.queue.popleft()
      return self.queue[0] if self.queue else -1

   def add(self, value):
      if value in self.counts:
          self.counts[value] += 1
      else:
          self.counts[value] = 1
          self.queue.append(value)
firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique())
firstUnique.add(5)
```

5. Valid string

```
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, arr, index):
        if not node or index == len(arr) or node.val != arr[index]:
            return False
        if index == len(arr) - 1:
            return not node.left and not node.right
        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, TreeNode(0)))
arr = [0, 1, 0, 1]
print(isValidSequence(root, arr))
```

6. Kids with greatest number of candies

```
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)
    return [(candy + extraCandies) >= max_candies for candy in candies]
print(kidsWithCandies([2, 3, 5, 1, 3], 3))
```

7. Max difference you can get from changing a number

```
def maxDifference(num):
    str_num = str(num)
    max_num = min_num = num
```

```
    for d in str_num:
        if d != '9':
            max_num = int(str_num.replace(d, '9'))
            break

    for d in str_num:
        if d != '1' and d != '0':
            min_num = int(str_num.replace(d, '1'))
            break

    return max_num - min_num

print(maxDifference(9))
```

8. Check if a string can break another string

```
def checkIfCanBreak(s1, s2):
    s1, s2 = sorted(s1), sorted(s2)
    return all(x >= y for x, y in zip(s1, s2)) or all(x <= y for x, y in zip(s1, s2))
print(checkIfCanBreak("abc", "xya"))
```

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

```
def number_ways_to_wear_hats(hats):
    MOD = 10**9 + 7
    n = len(hats)
```

```
    hat_to_person = {}
    for person, hats_list in enumerate(hats):
        for hat in hats_list:
            if hat not in hat_to_person:
                hat_to_person[hat] = []
            hat_to_person[hat].append(person)
    dp = [0] * (1 << n)
    dp[0] = 1
    for hat in range(1, 41):
        if hat in hat_to_person:
            for mask in range((1 << n) - 1, -1, -1):
                for person in hat_to_person[hat]:
                    if mask & (1 << person) == 0:
                        dp[mask | (1 << person)] = (dp[mask | (1 << person)] +
dp[mask]) % MOD
        return dp[(1 << n) - 1]


hats = [[3,4],[4,5],[5]]
print(number_ways_to_wear_hats(hats))
```

10. Next permutation

```
def destination_city(paths):
    starting_cities = set()
    for start, end in paths:
        starting_cities.add(start)

    for start, end in paths:
        if end not in starting_cities:
```

```
        return end

paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]
print(destination_city(paths))
```