Assignment-3

Name:G.venkata Praveen

Reg.no:192373023

Dept:cse(D.s)

1. Counting Elements

def count\_elements(arr):

count = 0

num\_set = set(arr)

for num in arr:

if num + 1 in num\_set:

count += 1

return count

# Example

arr = [1, 2, 3]

result = count\_elements(arr)

print(result) # Output: 2

A screenshot of a computer

Description automatically generated

1. Perform String Shifts

def string\_shift(s, shift):

total\_shift = 0

for direction, amount in shift:

if direction == 0:

total\_shift -= amount

else:

total\_shift += amount

total\_shift %= len(s)

return s[-total\_shift:] + s[:-total\_shift]

# Test the function with the provided example

s = "abc"

shift = [[0, 1], [1, 2]]

output = string\_shift(s, shift)

print(output) # Output: "cab"

A screenshot of a computer

Description automatically generated

1. Leftmost Column with at Least a One

class Solution:

def leftMostColumnWithOne(self, binaryMatrix: 'BinaryMatrix') -> int:

rows, cols = binaryMatrix.dimensions()

current\_row = 0

current\_col = cols - 1

leftmost\_col = -1

while current\_row < rows and current\_col >= 0:

if binaryMatrix.get(current\_row, current\_col) == 1:

leftmost\_col = current\_col

current\_col -= 1

else:

current\_row += 1

return leftmost\_col

1. First Unique Number

from collections import OrderedDict

class FirstUnique:

def \_\_init\_\_(self, nums):

self.queue = []

self.unique\_dict = OrderedDict()

for num in nums:

self.add(num)

def showFirstUnique(self):

if self.unique\_dict:

return next(iter(self.unique\_dict.values()))

return -1

def add(self, value):

if value in self.unique\_dict:

self.unique\_dict.pop(value)

elif value not in self.queue:

self.unique\_dict[value] = value

self.queue.append(value)

# Example Usage

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

A screenshot of a computer

Description automatically generated

5.check if a string valid or not

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 check\_path(node, 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 check\_path(node.left, index + 1) or check\_path(node.right, index + 1)

return check\_path(root, 0)

# Example Usage

root = TreeNode(0)

root.left = TreeNode(1)

root.right = TreeNode(0)

root.left.left = TreeNode(0)

root.left.right = TreeNode(1)

root.right.left = None

root.right.right = None

arr = [0, 1, 0, 1]

print(is\_valid\_sequence(root, arr)) # Output: True

A screenshot of a computer

Description automatically generated

6.Kids With the Greatest Number of Candies

def distribute\_candies(candies, extra\_candies):

try:

max\_candies = max(candies)

result = [candy + extra\_candies >= max\_candies for candy in candies]

return result

except Exception as e:

print(f"An error occurred: {e}")

return None

# Example

candies = [2, 3, 5, 1, 3]

extra\_candies = 3

output = distribute\_candies(candies, extra\_candies)

print(output)

A black background with white text

Description automatically generated

7.Max Difference You Can Get From Changing an Integer

def max\_diff(num):

def replace\_digit(n, x, y):

return int(str(n).replace(str(x), str(y)))

max\_diff\_result = 0

for x in range(10):

for y in range(10):

a = replace\_digit(num, x, 9)

b = replace\_digit(a, y, 1)

if a != 0 and b != 0 and a - b > max\_diff\_result:

max\_diff\_result = a - b

return max\_diff\_result

# Test the function with the provided example

num = 555

output = max\_diff(num)

print(output) # Output: 888

A screenshot of a computer

Description automatically generated

8. Check If a String Can Break Another String

def check\_permutation\_break(s1, s2):

if len(s1) != len(s2):

raise ValueError("Both strings must be of the same length.")

s1\_sorted = sorted(s1)

s2\_sorted = sorted(s2)

if all(s1\_char >= s2\_char for s1\_char, s2\_char in zip(s1\_sorted, s2\_sorted)) or all(s2\_char >= s1\_char for s1\_char, s2\_char in zip(s1\_sorted, s2\_sorted)):

return True

else:

return False

# Test the function with the provided example

s1 = "abc"

s2 = "xya"

result = check\_permutation\_break(s1, s2)

print(result) # Output: True

A screenshot of a computer

Description automatically generated

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

def num\_ways\_to\_choose\_hats(hats):

MOD = 10\*\*9 + 7

n = len(hats)

all\_hats = 1 << 40

dp = [0] \* all\_hats

dp[0] = 1

for i in range(1, n + 1):

for hat in hats[i - 1]:

for j in range(all\_hats - 1, -1, -1):

if j & (1 << hat):

dp[j] += dp[j ^ (1 << hat)]

dp[j] %= MOD

return dp[-1]

# Example

hats = [[3, 4], [4, 5], [5]]

try:

result = num\_ways\_to\_choose\_hats(hats)

print("Number of ways to choose hats:", result)

except Exception as e:

print("An error occurred:", e)

10. Next Permutation

def next\_permutation(nums):

# Find the first decreasing element from the right

i = len(nums) - 2

while i >= 0 and nums[i] >= nums[i + 1]:

i -= 1

if i >= 0:

# Find the next greater element to swap with

j = len(nums) - 1

while nums[j] <= nums[i]:

j -= 1

nums[i], nums[j] = nums[j], nums[i]

# Reverse the remaining elements

nums[i + 1:] = nums[i + 1:][::-1]

# Test the function with the given example

nums = [1, 2, 3]

print("Input:", nums)

next\_permutation(nums)

print("Output:", nums)

A screenshot of a computer

Description automatically generated