1.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.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.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.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.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.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.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.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.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.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))