**ASSIGNMENT**

**1. Odd String Difference You are given an array of equal-length strings words. Assume that the length of each string is n.Each string words[i] can be converted into a difference integer array difference[i] of length n - 1 where difference[i][j] = words[i][j+1] - words[i][j] where 0 <= j <= n - 2. Note that the difference between two letters is the difference between their positions in the alphabet i.e. the position of 'a' is 0, 'b' is 1, and 'z' is 25. For example, for the string "acb", the difference integer array is [2 - 0, 1 - 2] = [2, -1].All the strings in words have the same difference integer array, except one. You should find that string. Return the string in words that has dif erent dif erence integer array. Example 1: Input: words = ["adc","wzy","abc"] Output: "abc" Explanation: - The difference integer array of "adc" is [3 - 0, 2 - 3] = [3, -1]. - The difference integer array of "wzy" is [25 - 22, 24 - 25]= [3, -1]. - The difference integer array of "abc" is [1 - 0, 2 - 1] = [1, 1]. The odd array out is [1, 1], so we return the corresponding string, "abc**

def find\_odd\_string(words):

def get\_difference\_array(word):

return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]

diff\_arrays = [get\_difference\_array(word) for word in words]

diff\_count = {}

for i, diff in enumerate(diff\_arrays):

diff\_tuple = tuple(diff)

if diff\_tuple in diff\_count:

diff\_count[diff\_tuple].append(i)

else:

diff\_count[diff\_tuple] = [i]

for diff\_tuple, indices in diff\_count.items():

if len(indices) == 1:

return words[indices[0]]

**2. Words Within Two Edits of Dictionary You are given two string arrays, queries and dictionary. All words in each array comprise of lowercase English letters and have the same length.In one edit you can take a word from queries, and change any letter in it to any other letter. Find all words from queries that, after a maximum of two edits, equal some word from dictionary. Return a list of all words from queries, that match with some word from dictionary after a maximum of two edits. Return the words in the same order they appear in queries. Example 1: Input: queries = ["word","note","ants","wood"], dictionary = ["wood","joke","moat"] Output: ["word","note","wood"] Explanation: - Changing the 'r' in "word" to 'o' allows it to equal the dictionary word "wood". - Changing the 'n' to 'j' and the 't' to 'k' in "note" changes it to "joke". - It would take more than 2 edits for "ants" to equal a dictionary word. - "wood" can remain unchanged (0 edits) and match the corresponding dictionary word. Thus, we return ["word","note","wood"].**def words\_within\_two\_edits(queries, dictionary):

def within\_two\_edits(word1, word2):

return sum(1 for x, y in zip(word1, word2) if x != y) <= 2

result = []

for query in queries:

for dict\_word in dictionary:

if within\_two\_edits(query, dict\_word):

result.append(query)

break

return result

**3. Destroy Sequential Targets You are given a 0-indexed array nums consisting of positive integers, representing targets on a number line. You are also given an integer space. You have a machine which can destroy targets. Seeding the machine with some nums[i] allows it to destroy all targets with values that can be represented as nums[i] + c \* space, where c is any non-negative integer. You want to destroy the maximum number of targets in nums. Return the minimum value of nums[i] you can seed the machine with to destroy the maximum number of targets. Example 1: Input: nums = [3,7,8,1,1,5], space = 2 Output: 1 Explanation: If we seed the machine with nums[3], then we destroy all targets equal to 1,3,5,7,9,... In this case, we would destroy 5 total targets (all except for nums[2]). It is impossible to destroy more than 5 targets, so we return nums[3].**

def destroy\_sequential\_targets(nums, space):

from collections import defaultdict

remainder\_groups = defaultdict(list)

for num in nums:

remainder = num % space

remainder\_groups[remainder].append(num)

max\_group\_size = 0

smallest\_num\_in\_max\_group = float('inf')

for remainder, group in remainder\_groups.items():

group\_size = len(group)

if group\_size > max\_group\_size:

max\_group\_size = group\_size

smallest\_num\_in\_max\_group = min(group)

elif group\_size == max\_group\_size:

smallest\_num\_in\_max\_group = min(smallest\_num\_in\_max\_group, min(group))

return smallest\_num\_in\_max\_group

**4. Next Greater Element IV You are given a 0-indexed array of non-negative integers nums. For each integer in nums, you must find its respective second greater integer. The second greater integer of nums[i] is nums[j] such that: ● j > i ● nums[j] > nums[i] ● There exists exactly one index k such that nums[k] > nums[i] and i < k < j. If there is no such nums[j], the second greater integer is considered to be -1. ● For example, in the array [1, 2, 4, 3], the second greater integer of 1 is 4, 2 is 3, and that of 3 and 4 is -1. Return an integer array answer, where answer[i] is the second greater integer of nums[i]. Example 1: Input: nums = [2,4,0,9,6] Output: [9,6,6,-1,-1] Explanation: 0th index: 4 is the first integer greater than 2, and 9 is the second integer greater than 2, to the right of 2. 1st index: 9 is the first, and 6 is the second integer greater than 4, to the right of 4. 2nd index: 9 is the first, and 6 is the second integer greater than 0, to the right of 0. 3rd index: There is no integer greater than 9 to its right, so the second greater integer is considered to be -1. 4th index: There is no integer greater than 6 to its right, so the second greater integer is considered to be -1. Thus, we return [9,6,6,-1,-1].**

def next\_greater\_element\_IV(nums):

n = len(nums)

result = [-1] \* n

first\_greater = []

second\_greater = []

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

while first\_greater and nums[first\_greater[-1]] <= nums[i]:

first\_greater.pop()

if first\_greater:

while second\_greater and nums[second\_greater[-1]] <= nums[i]:

second\_greater.pop()

if second\_greater:

result[i] = nums[second\_greater[-1]]

if first\_greater:

second\_greater.append(first\_greater[-1])

first\_greater.append(i)

return result

**5. Average Value of Even Numbers That Are Divisible by Three Given an integer array nums of positive integers, return the average value of all even integers that are divisible by 3. Note that the average of n elements is the sum of the n elements divided by n and rounded down to the nearest integer. Example 1: Input: nums = [1,3,6,10,12,15] Output: 9 Explanation: 6 and 12 are even numbers that are divisible by 3. (6 + 12) / 2 = 9.**

def average\_value\_of\_even\_divisible\_by\_three(nums):

filtered\_nums = [num for num in nums if num % 6 == 0] # % 6 ensures the number is both even and divisible by 3

total\_sum = sum(filtered\_nums)

count = len(filtered\_nums)

if count == 0:

return 0 # To handle the case where there are no numbers satisfying the condition

average = total\_sum // count

return average

**6. Most Popular Video Creator You are given two string arrays creators and ids, and an integer array views, all of length n. The ith video on a platform was created by creator[i], has an id of ids[i], and has views[i] views. The popularity of a creator is the sum of the number of views on all of the creator's videos. Find the creator with the highest popularity and the id of their most viewed video. ● If multiple creators have the highest popularity, find all of them. ● If multiple videos have the highest view count for a creator, find the lexicographically smallest id. Return a 2D array of strings answer where answer[i] = [creatori, idi] means that creatori has the highest popularity and idi is the id of their most popular video. The answer can be returned in any order. Example 1: Input: creators = ["alice","bob","alice","chris"], ids = ["one","two","three","four"], views = [5,10,5,4] Output: [["alice","one"],["bob","two"]] Explanation: The popularity of alice is 5 + 5 = 10. The popularity of bob is 10. The popularity of chris is 4. alice and bob are the most popular creators. For bob, the video with the highest view count is "two". For alice, the videos with the highest view count are "one" and "three". Since "one" is lexicographically smaller than "three", it is included in the answer.**

def mostPopularVideoCreator(creators, ids, views):

creator\_popularity = {}

most\_viewed\_video = {}

for creator, video\_id, view in zip(creators, ids, views):

if creator not in creator\_popularity:

creator\_popularity[creator] = 0

most\_viewed\_video[creator] = (view, video\_id)

creator\_popularity[creator] += view

if (view > most\_viewed\_video[creator][0] or

(view == most\_viewed\_video[creator][0] and video\_id < most\_viewed\_video[creator][1])):

most\_viewed\_video[creator] = (view, video\_id)

max\_popularity = max(creator\_popularity.values())

result = []

for creator in creator\_popularity:

if creator\_popularity[creator] == max\_popularity:

result.append([creator, most\_viewed\_video[creator][1]])

return result

**7. Minimum Addition to Make Integer Beautiful You are given two positive integers n and target. An integer is considered beautiful if the sum of its digits is less than or equal to target. Return the minimum non-negative integer x such that n + x is beautiful. The input will be generated such that it is always possible to make n beautiful. Example 1: Input: n = 16, target = 6 Output: 4 Explanation: Initially n is 16 and its digit sum is 1 + 6 = 7. After adding 4, n becomes 20 and digit sum becomes 2 + 0 = 2. It can be shown that we can not make n beautiful with adding non-negative integer less than 4**.

def sum\_of\_digits(num):

total = 0

while num > 0:

total += num % 10

num //= 10

return total

def min\_addition\_to\_make\_beautiful(n, target):

x = 0

while sum\_of\_digits(n + x) > target:

x += 1

return x

**8. Split Message Based on Limit You are given a string, message, and a positive integer, limit. You must split message into one or more parts based on limit. Each resulting part should have the suffix "", where "b" is to be replaced with the total number of parts and "a" is to be replaced with the index of the part, starting from 1 and going up to b. Additionally, the length of each resulting part (including its suffix) should be equal to limit, except for the last part whose length can be at most limit. The resulting parts should be formed such that when their suffixes are removed and they are all concatenated in order, they should be equal to message. Also, the result should contain as few parts as possible. Return the parts message would be split into as an array of strings. If it is impossible to split message as required, return an empty array. Example 1: Input: message = "this is really a very awesome message", limit = 9 Output: ["thi","s i","s r","eal","ly ","a v","ery"," aw","eso","me"," m","es","sa","ge"] Explanation: The first 9 parts take 3 characters each from the beginning of message. The next 5 parts take 2 characters each to finish splitting message. In this example, each part, including the last, has length 9. It can be shown it is not possible to split message into less than 14 parts.**

def split\_message(message, limit):

def calculate\_suffix\_length(parts):

total\_digits = len(str(parts))

return total\_digits \* 2 + 3 # The length of '<a/b>'

def can\_split\_into\_parts(parts):

suffix\_length = calculate\_suffix\_length(parts)

if suffix\_length >= limit:

return False

part\_size = limit - suffix\_length

total\_length\_needed = len(message) + parts \* suffix\_length

return total\_length\_needed <= parts \* limit

low, high = 1, len(message)

while low < high:

mid = (low + high) // 2

if can\_split\_into\_parts(mid):

high = mid

else:

low = mid + 1

total\_parts = low

if not can\_split\_into\_parts(total\_parts):

return []

result = []

idx = 0

for part in range(1, total\_parts + 1):

suffix = f"<{part}/{total\_parts}>"

part\_size = limit - len(suffix)

part\_message = message[idx:idx + part\_size]

result.append(part\_message + suffix)

idx += part\_size

return result