ASSIGNMENT - 4

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.

**CODE**:

def odd\_string\_difference(words):

# Helper function to convert a string to its difference array

def to\_difference\_array(word):

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

# Convert all words to their difference arrays

difference\_arrays = [to\_difference\_array(word) for word in words]

# Use a dictionary to count the occurrences of each difference array difference\_count = {}

for diff\_array in difference\_arrays:

diff\_tuple = tuple(diff\_array) # Convert list to tuple to use as dict key if diff\_tuple in difference\_count:

difference\_count[diff\_tuple] += 1

else:

difference\_count[diff\_tuple] = 1

# Find the difference array that occurs only once

for diff\_array in difference\_arrays:

if difference\_count[tuple(diff\_array)] == 1:

odd\_diff\_array = diff\_array

break

# Find and return the word corresponding to the odd difference array

for word in words:

if to\_difference\_array(word) == odd\_diff\_array:

return word

# Example usage:

words = ["abc", "bcd", "ace"]

print(odd\_string\_difference(words))

**OUTPUT**:



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"].

**CODE**:

def words\_within\_two\_edits(queries, dictionary):

# Helper function to check if two words differ by at most two characters

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

# Check if the two words differ by at most two characters

count\_diff = sum(1 for a, b in zip(word1, word2) if a != b)

return count\_diff <= 2

# List to store the results

result = []

# ChEck each word in queries against each word in dictionary

for query in queries:

for dict\_word in dictionary:

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

result.append(query)

break

return result

# Example usage:

queries = ["word", "note", "ants", "wood"]

dictionary = ["wood", "joke", "moat"]

print(words\_within\_two\_edits(queries, dictionary))

**OUTPUT:**



3. 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].

**CODE:**

def second\_greater\_element(nums):

# Initialize the result array with -1 for each element

result = [-1] \* len(nums)

# Iterate through the array to find the second greater element for each nums[i] for i in range(len(nums)):

first\_greater\_found = False

for j in range(i + 1, len(nums)):

if nums[j] > nums[i]:

if not first\_greater\_found:

first\_greater\_found = True

else:

result[i] = nums[j]

break

return result

# Example usage:

nums = [1, 2, 4, 3]

print(second\_greater\_element(nums))

**OUTPUT**:



4. 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.

**CODE:**

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

# Helper function to calculate the sum of digits of a number

def sum\_of\_digits(num):

return sum(int(digit) for digit in str(num))

# If the sum of digits of n is already <= target, no addition is needed

if sum\_of\_digits(n) <= target:

return 0

# Initialize the result x to 0

x = 0

increment = 1

# Process each digit from the least significant to the most significant

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

# Calculate the next multiple of 10 for the least significant digit position next\_increment = increment - (n % increment)

x += next\_increment

n += next\_increment

increment \*= 10

return x

# Example usage:

n = 467

target = 15

print(min\_addition\_to\_make\_beautiful(n, target))

**OUTPUT:**



5. Sort Array by Moving Items to Empty Space

You are given an integer array nums of size n containing each element from 0 to n - 1 (inclusive). Each of the elements from 1 to n - 1 represents an item, and the element 0 represents an empty space.

In one operation, you can move any item to the empty space. nums is considered to be sorted if the numbers of all the items are in ascending order and the empty space is either at the beginning or at the end of the array.

For example, if n = 4, nums is sorted if:

● nums = [0,1,2,3] or

● nums = [1,2,3,0]

...and considered to be unsorted otherwise.

Return the minimum number of operations needed to sort nums.

**CODE:**

def min\_operations\_to\_sort(nums):

n = len(nums)

target1 = list(range(n)) # [0, 1, 2, ..., n-1]

target2 = list(range(1, n)) + [0] # [1, 2, ..., n-1, 0]

def count\_moves(target):

nums\_copy = nums[:]

pos = {num: i for i, num in enumerate(nums\_copy)} # positions of each number

moves = 0

for i in range(n):

while nums\_copy[i] != target[i]:

empty\_index = pos[0]

target\_num\_index = pos[target[i]]

# Swap the element at target\_num\_index with the empty space

nums\_copy[empty\_index], nums\_copy[target\_num\_index] = nums\_copy[target\_num\_index], nums\_copy[empty\_index]

# Update positions in the map

pos[nums\_copy[empty\_index]] = empty\_index pos[nums\_copy[target\_num\_index]] = target\_num\_index

moves += 1

return moves

# Compute moves for both possible target configurations return min(count\_moves(target1), count\_moves(target2))

# Example usage:

nums = [2, 0, 1, 3]

print(min\_operations\_to\_sort(nums))

OUTPUT:

