ASSIGNMENT-4

K.SAISOMASEKHAR REDDY

192324134

DEPT.:CSE(AI&DS)

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 different difference 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".

Example 2:

Input: words = ["aaa","bob","ccc","ddd"]

Output: "bob"

Explanation: All the integer arrays are [0, 0] except for "bob", which corresponds to [13, -13].

Constraints:

3 <= words.length <= 100

n == words[i].length

2 <= n <= 20

words[i] consists of lowercase English letters

PROGRAM:

```
def find_odd_string(words):
    def calculate_difference(word):
        return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]

    difference_arrays = [calculate_difference(word) for word in words]


    unique_difference = None
    for diff in difference_arrays:
        if difference_arrays.count(diff) == 1:
            unique_difference = diff
            break

    for i, diff in enumerate(difference_arrays):
        if diff == unique_difference:
            return words[i]

    return None

# Example usage:
words1 = ["adc", "wzy", "abc"]
print("Example 1 Output:", find_odd_string(words1))

words2 = ["aaa", "bob", "ccc", "ddd"]
print("Example 2 Output:", find_odd_string(words2))
```

OUTPUT:

```
Example 1 Output: abc
Example 2 Output: bob
```

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"].

Example 2:

Input: queries = ["yes"], dictionary = ["not"]

Output: []

Explanation:

Applying any two edits to "yes" cannot make it equal to "not". Thus, we return an empty array.

Constraints:

1 <= queries.length, dictionary.length <= 100

n == queries[i].length == dictionary[j].length

1 <= n <= 100

All queries[i] and dictionary[j] are composed of lowercase English letters.

2453. 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].

Example 2:

Input: nums = [1,3,5,2,4,6], space = 2

Output: 1

Explanation: Seeding the machine with nums[0], or nums[3] destroys 3 targets.

It is not possible to destroy more than 3 targets.

Since nums[0] is the minimal integer that can destroy 3 targets, we return 1.

Example 3:

Input: nums = [6,2,5], space = 100

Output: 2

Explanation: Whatever initial seed we select, we can only destroy 1 target. The minimal seed is nums[1].

Constraints:

1 <= nums.length <= 105

1 <= nums[i] <= 109

1 <= space <= 109

PROGRAM:

```
def min_seed_value(nums, space):
    max_destroyed = 0
    min_seed = float('inf')

    for num in nums:
        destroyed = (num - 1) // space
        if destroyed >= max_destroyed:
            max_destroyed = destroyed
            min_seed = min(min_seed, num)
    return min_seed
# Example usage:
nums1 = [3, 7, 8, 1, 1, 5]
space1 = 2
print("Example 1 Output:", min_seed_value(nums1, space1))

nums2 = [1, 3, 5, 2, 4, 6]
space2 = 2
print("Example 2 Output:", min_seed_value(nums2, space2))

nums3 = [6, 2, 5]
space3 = 100
print("Example 3 Output:", min_seed_value(nums3, space3))
```

OUTPUT:

```
Example 1 Output: 3
Example 2 Output: 1
Example 3 Output: 2
```

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].

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].

Example 2:

Input: nums = [3,3]

Output: [-1,-1]

Explanation:

We return [-1,-1] since neither integer has any integer greater than it.

Constraints:

1 <= nums.length <= 105

0 <= nums[i] <= 109

2455. 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.
PROGRAM:

```
def next_greater_element(nums):
    stack = []
    result = [-1] * len(nums)

    for i in range(len(nums)):
        while stack and nums[i] > nums[stack[-1]]:
            result[stack.pop()] = nums[i]
        stack.append(i)

    return result

# Example usage:
nums1 = [2, 4, 0, 9, 6]
print("Example 1 Output:", next_greater_element(nums1))

nums2 = [3, 3]
print("Example 2 Output:", next_greater_element(nums2))
```

OUTPUT:

```
Example 1 Output: [4, 9, 9, -1, -1]
Example 2 Output: [-1, -1]
```

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.

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.

Example 2:

Input: n = 467, target = 6

Output: 33

Explanation: Initially n is 467 and its digit sum is 4 + 6 + 7 = 17. After adding 33, n becomes 500 and digit sum becomes 5 + 0 + 0 = 5. It can be shown that we can not make n beautiful with adding non-negative integer less than 33.

Example 3:

Input: n = 1, target = 1

Output: 0

Explanation: Initially n is 1 and its digit sum is 1, which is already smaller than or equal to target.

Constraints:

- 1 <= n <= 1012
- 1 <= target <= 150
- The input will be generated such that it is always possible to make n beautiful.

PROGRAM:

```
def digit_sum(num):
    # Calculate the sum of digits in a number
    return sum(int(digit) for digit in str(num))

def min_beautiful_integer(n, target):
    current_sum = digit_sum(n)
    x = max(0, target - current_sum)

    return x

# Example usage:
n1, target1 = 16, 6
print("Example 1 Output:", min_beautiful_integer(n1, target1))

n2, target2 = 467, 6
print("Example 2 Output:", min_beautiful_integer(n2, target2))

n3, target3 = 1, 1
print("Example 3 Output:", min_beautiful_integer(n3, target3))
```

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.

Example 1:

Input: nums = [4,2,0,3,1]

Output: 3

Explanation:

- Move item 2 to the empty space. Now, nums = [4,0,2,3,1].

- Move item 1 to the empty space. Now, nums = [4,1,2,3,0].

- Move item 4 to the empty space. Now, nums = [0,1,2,3,4].

It can be proven that 3 is the minimum number of operations needed.

PROGRAM:

```
def min_operations_to_sort(nums):
    n = len(nums)
    visited = [False] * n
    swaps = 0

    for i in range(n):
        if not visited[i]:
            visited[i] = True
            j = nums[i]
            while not visited[j]:
                visited[j] = True
                j = nums[j]
                swaps += 1

    return swaps

# Example usage:
nums = [4, 2, 0, 3, 1]
print("Output:", min operations to sort(nums))   # Output: 3
```

OUTPUT:

```
==== RESTAR
Output: 3
```