DATE:10/06/2024

1. Odd String Difference

You are given an array of equal-length strings words. Assume that the length of each string is n.

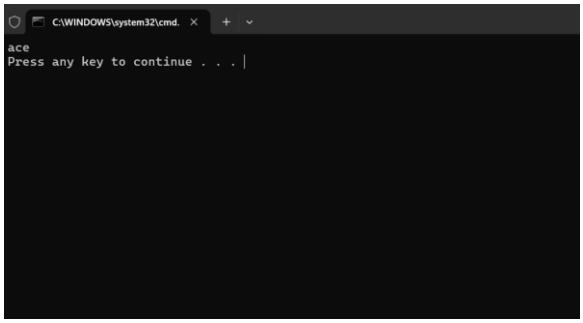
Each string words[i] can be converted into a difference integer array difference[i] of length n - 1 where difference[i][j] = words[i][j+1] - words[i][j] where 0 <= j <= n - 2.

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

def odd_string_difference(words):
    difference_arrays = [to_difference_array(word) for word in words]

    difference_count = {}
    for diff_array in difference_arrays:
        diff_tuple = tuple(diff_array)
        if diff_tuple in difference_count:
            difference_count[diff_tuple] += 1
        else:
            difference_count[diff_tuple] = 1

    for diff_array in difference_arrays:
        if difference_count[tuple(diff_array)] == 1:
            odd_diff_array = diff_array
            break

    for word in words:
        if to_difference_array(word) == odd_diff_array:
            return word

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):
    def within_two_edits(word1, word2):
        count_diff = sum(1 for a, b in zip(word1, word2) if a != b)
        return count_diff <= 2

    result = []

    for query in queries:
        for dict_word in dictionary:
            if within_two_edits(query, dict_word):
                result.append(query)
                break

    return result

queries = ["word", "note", "ants", "wood"]
dictionary = ["wood", "joke", "moat"]
print(words_within_two_edits(queries, dictionary))
```

output:

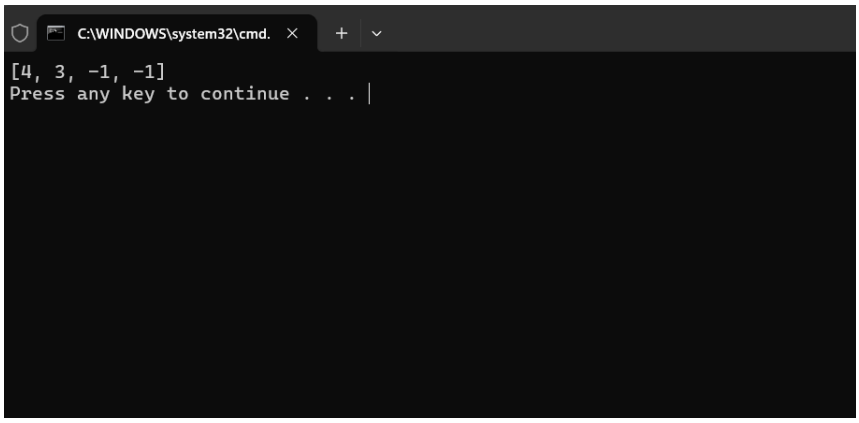
3. Next Greater Element IV
You are given a 0-indexed array of non-negative integers nums. For each integer in nums, you must find its respective second greater integer.
The second greater integer of nums[i] is nums[j] such that:
j > i
nums[j] > nums[i]

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


    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

nums = [1, 2, 4, 3]
print(second_greater_element(nums))
```

OUTPUT:

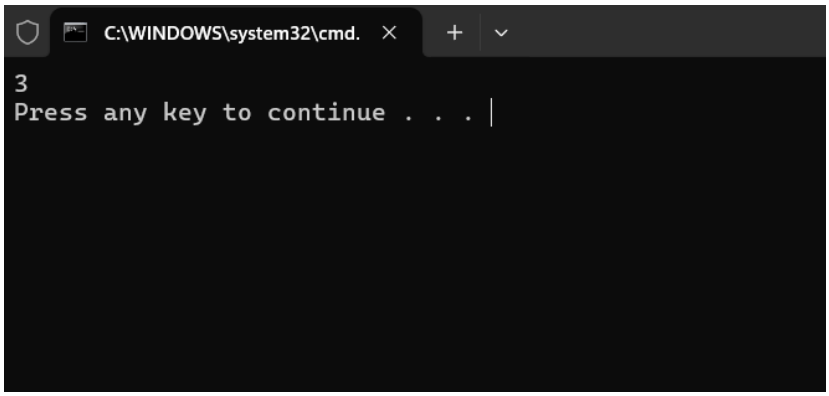


4. Minimum Addition to Make Integer Beautiful
You are given two positive integers n and target.
An integer is considered beautiful if the sum of its digits is less than or equal to target. Return the minimum non-negative integer x such that n + x is beautiful. The input will be
generated such that it 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 should be 533
```

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]

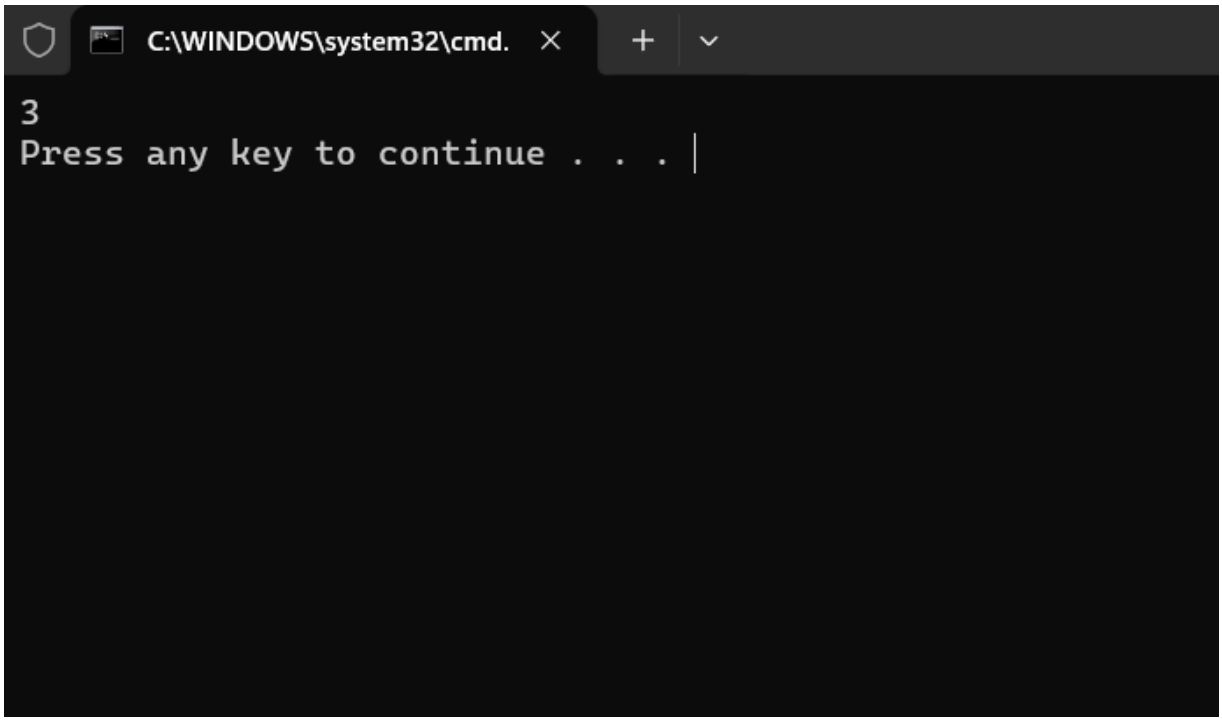
...and considered to be unsorted otherwise.

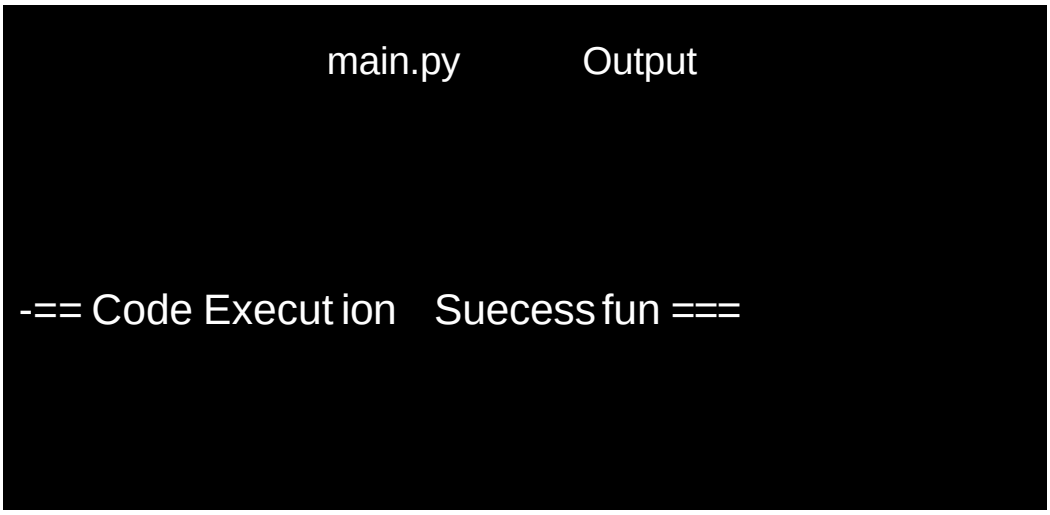
Return the minimum number of operations needed to sort nums.

CODE:

```
def min_operations_to_sort(nums):
    n = len(nums)
    target1 = list(range(n))
    target2 = list(range(1, n)) + [0]

    def count_moves(target):
        nums_copy = nums[:]
        pos = {num: i for i, num in enumerate(nums_copy)}

        moves = 0

        for i in range(n):
            while nums_copy[i] != target[i]:
                empty_index = pos[0]
                target_num_index = pos[target[i]]

                nums_copy[empty_index], nums_copy[target_num_index] =
nums_copy[target_num_index], nums_copy[empty_index]

                pos[nums_copy[empty_index]] = empty_index
                pos[nums_copy[target_num_index]] = target_num_index

                moves += 1

        return moves

    return min(count_moves(target1), count_moves(target2))

nums = [2, 0, 1, 3]

 print(min_operations_to_sort(nums))
```

OUTPUT:


C:\WINDOWS\system32\cmd.
3
Press any key to continue . . .


main.py
Output
-== Code Execut ion Suecess fun ===