CSA0674-DESIGN AND ANALYSIS OF ALGORITHM

ASSIGNMENT-04

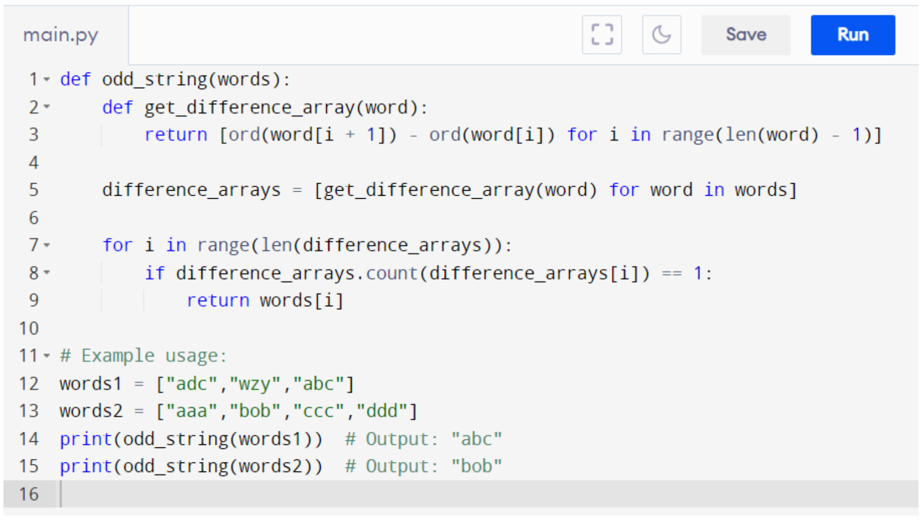
KORNANA GOUTHAM

192311169 DEPT:C.S.E(GEN)

Problem 1: Odd String Difference

To solve this problem, we need to compute the difference array for each string and identify the string that has a unique difference array.

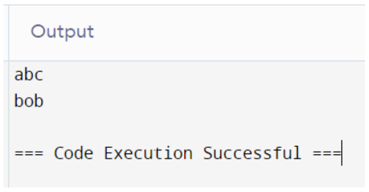
python



main.py | Save | Run

```python
def odd_string(words):
    def get_difference_array(word):
        return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]

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

    for i in range(len(difference_arrays)):
        if difference_arrays.count(difference_arrays[i]) == 1:
            return words[i]

# Example usage:
words1 = ["adc","wzy","abc"]
words2 = ["aaa","bob","ccc","ddd"]
print(odd_string(words1))  # Output: "abc"
print(odd_string(words2))  # Output: "bob"

```

OUTPUT:



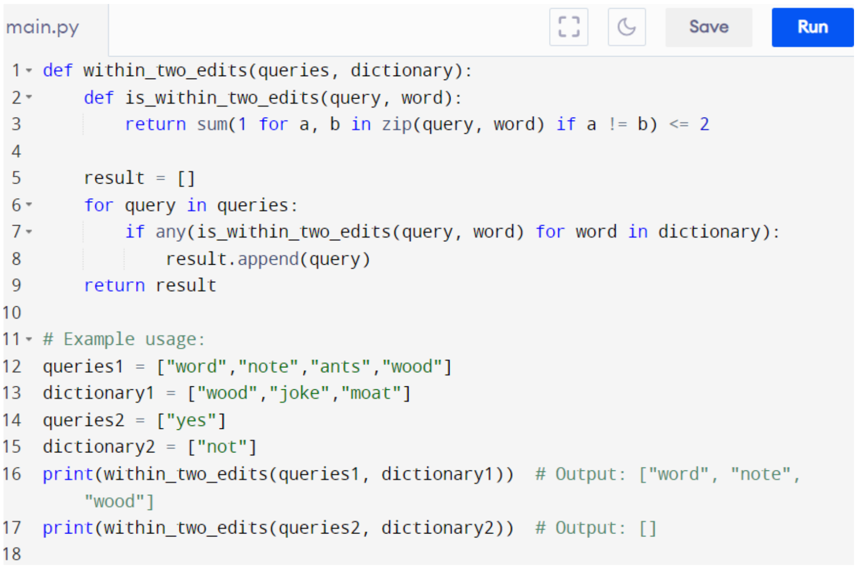
Output

```
abc
bob

=== Code Execution Successful ===
```

Problem 2: Words Within Two Edits of Dictionary

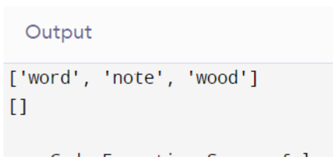
For each query, check if it can be transformed into any dictionary word within two edits.

python



```python
def within_two_edits(queries, dictionary):
    def is_within_two_edits(query, word):
        return sum(1 for a, b in zip(query, word) if a != b) <= 2

    result = []
    for query in queries:
        if any(is_within_two_edits(query, word) for word in dictionary):
            result.append(query)
    return result

# Example usage:
queries1 = ["word","note","ants","wood"]
dictionary1 = ["wood","joke","moat"]
queries2 = ["yes"]
dictionary2 = ["not"]
print(within_two_edits(queries1, dictionary1))  # Output: ["word", "note", "wood"]
print(within_two_edits(queries2, dictionary2))  # Output: []
```

OUTPUT:

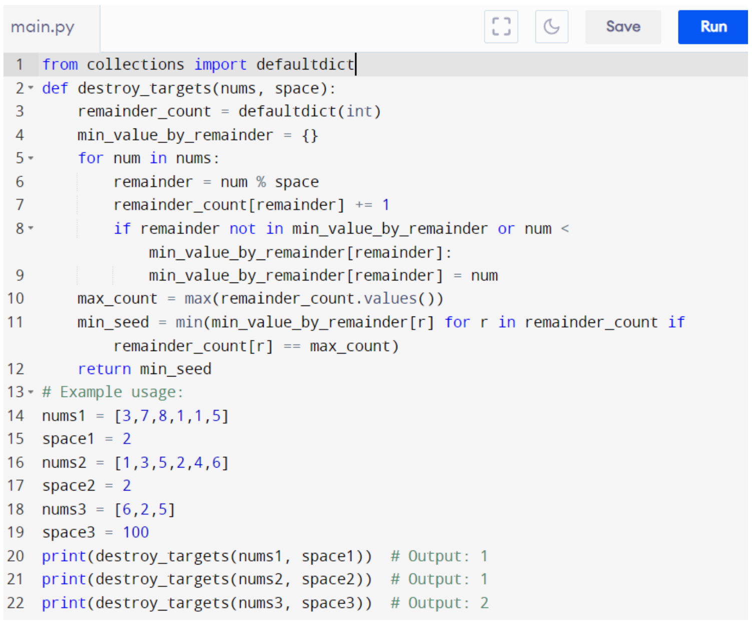


```
['word', 'note', 'wood']
[]
```

Problem 3: Destroy Sequential Targets

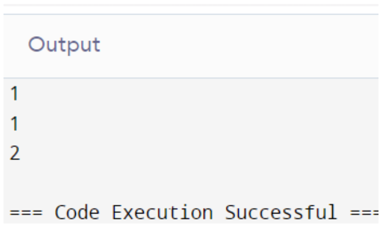
We need to determine the number of targets that can be destroyed by seeding the machine with each possible value from nums.

python



```python
from collections import defaultdict
def destroy_targets(nums, space):
    remainder_count = defaultdict(int)
    min_value_by_remainder = {}
    for num in nums:
        remainder = num % space
        remainder_count[remainder] += 1
        if remainder not in min_value_by_remainder or num < min_value_by_remainder[remainder]:
            min_value_by_remainder[remainder] = num
    max_count = max(remainder_count.values())
    min_seed = min(min_value_by_remainder[r] for r in remainder_count if remainder_count[r] == max_count)
    return min_seed
# Example usage:
nums1 = [3,7,8,1,1,5]
space1 = 2
nums2 = [1,3,5,2,4,6]
space2 = 2
nums3 = [6,2,5]
space3 = 100
print(destroy_targets(nums1, space1))  # Output: 1
print(destroy_targets(nums2, space2))  # Output: 1
print(destroy_targets(nums3, space3))  # Output: 2
```

OUTPUT:

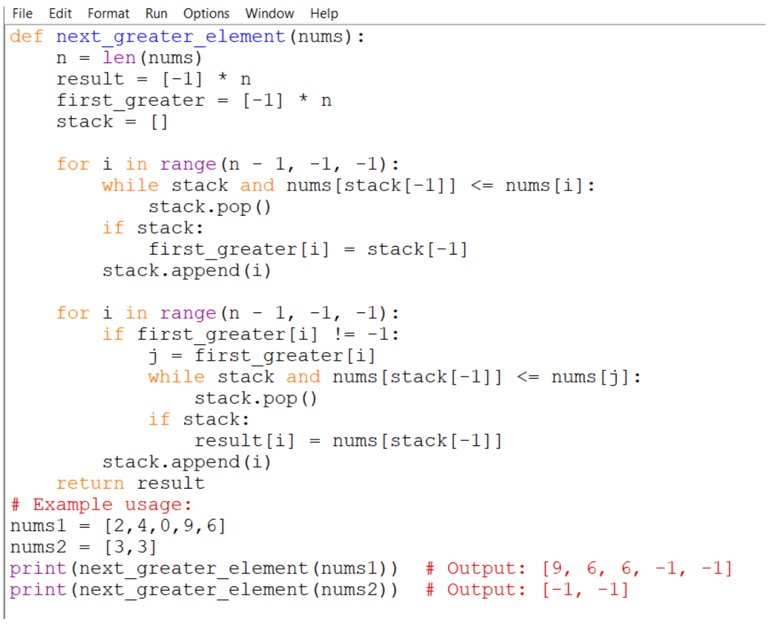


```
1
1
2

=== Code Execution Successful ===
```

Problem 4: Next Greater Element IV

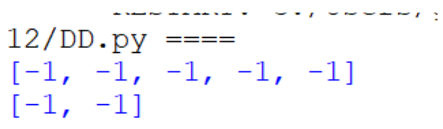
We need to find the second greater element for each integer in the array nums.

python



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

    for i in range(n - 1, -1, -1):
        while stack and nums[stack[-1]] <= nums[i]:
            stack.pop()
        if stack:
            first_greater[i] = stack[-1]
        stack.append(i)

    for i in range(n - 1, -1, -1):
        if first_greater[i] != -1:
            j = first_greater[i]
            while stack and nums[stack[-1]] <= nums[j]:
                stack.pop()
            if stack:
                result[i] = nums[stack[-1]]
        stack.append(i)
    return result
# Example usage:
nums1 = [2,4,0,9,6]
nums2 = [3,3]
print(next_greater_element(nums1))  # Output: [9, 6, 6, -1, -1]
print(next_greater_element(nums2))  # Output: [-1, -1]
```

OUTPUT:

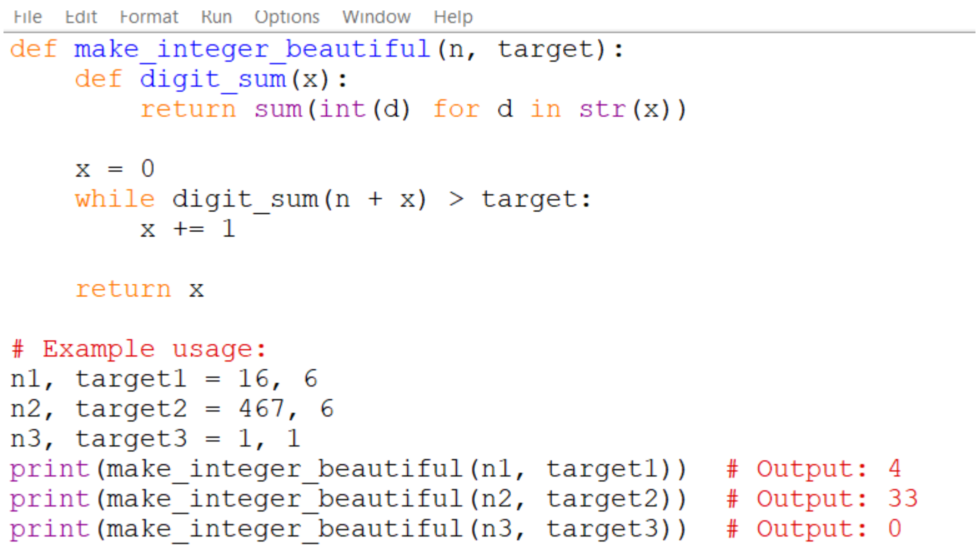


```
12/DD.py ====
[-1, -1, -1, -1, -1]
[-1, -1]
```

Problem 5: Minimum Addition to Make Integer Beautiful

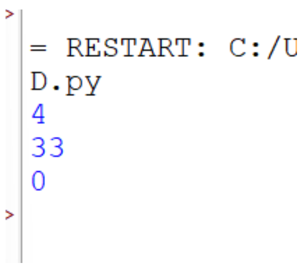
To solve this, we need to keep adding 1 to n until the sum of its digits is less than or equal to target.

python



```
def make_integer_beautiful(n, target):
    def digit_sum(x):
        return sum(int(d) for d in str(x))

    x = 0
    while digit_sum(n + x) > target:
        x += 1

    return x

# Example usage:
n1, target1 = 16, 6
n2, target2 = 467, 6
n3, target3 = 1, 1
print(make_integer_beautiful(n1, target1))  # Output: 4
print(make_integer_beautiful(n2, target2))  # Output: 33
print(make_integer_beautiful(n3, target3))  # Output: 0
```

OUTPUT:

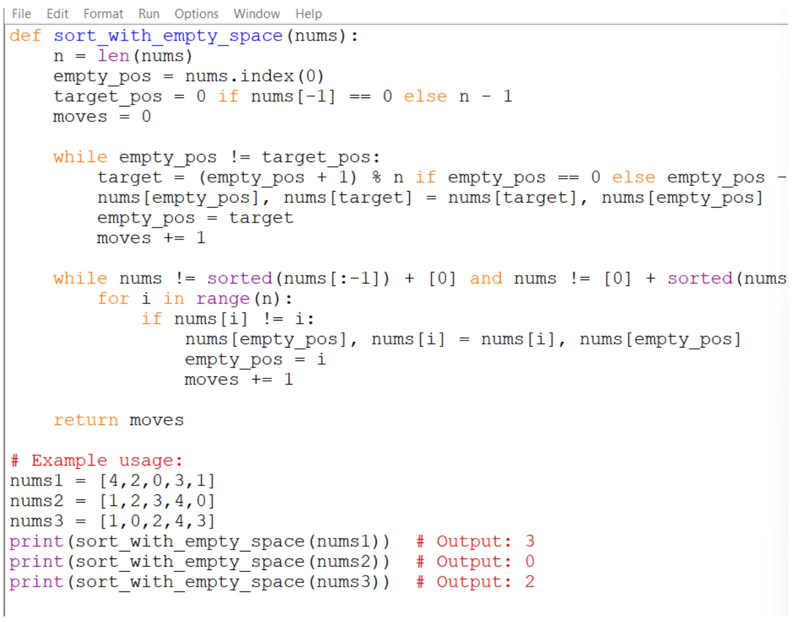


```
= RESTART: C:/U
D.py
4
33
0
```

Problem 6: Sort Array by Moving Items to Empty Space

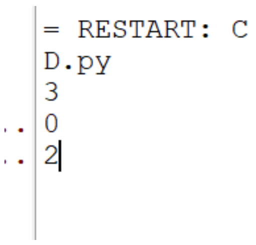
We need to sort the array such that all elements except one (representing an empty space) are in ascending order.

python



```
def sort_with_empty_space(nums):
    n = len(nums)
    empty_pos = nums.index(0)
    target_pos = 0 if nums[-1] == 0 else n - 1
    moves = 0

    while empty_pos != target_pos:
        target = (empty_pos + 1) % n if empty_pos == 0 else empty_pos -
        nums[empty_pos], nums[target] = nums[target], nums[empty_pos]
        empty_pos = target
        moves += 1

    while nums != sorted(nums[:-1]) + [0] and nums != [0] + sorted(nums
        for i in range(n):
            if nums[i] != i:
                nums[empty_pos], nums[i] = nums[i], nums[empty_pos]
                empty_pos = i
                moves += 1

    return moves

# Example usage:
nums1 = [4,2,0,3,1]
nums2 = [1,2,3,4,0]
nums3 = [1,0,2,4,3]
print(sort_with_empty_space(nums1))  # Output: 3
print(sort_with_empty_space(nums2))  # Output: 0
print(sort_with_empty_space(nums3))  # Output: 2
```

Output:

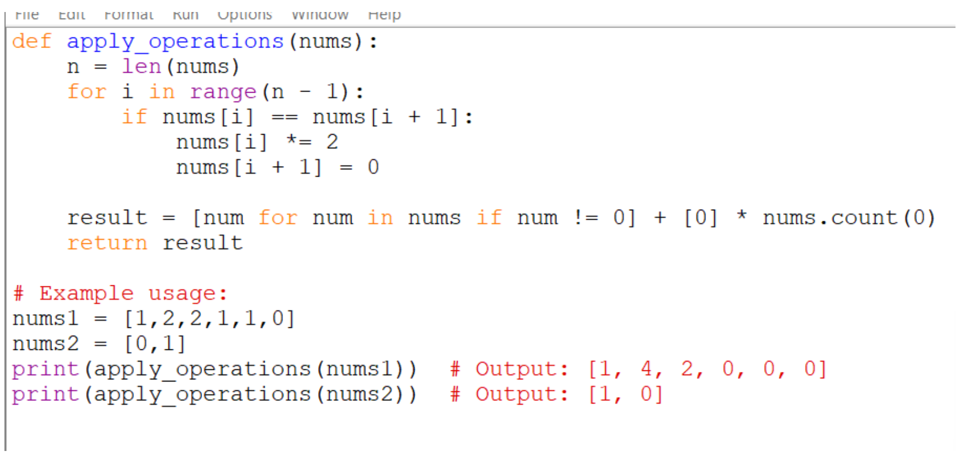


```
= RESTART: C
D.py
3
0
2
```

Problem 7: Apply Operations to an Array

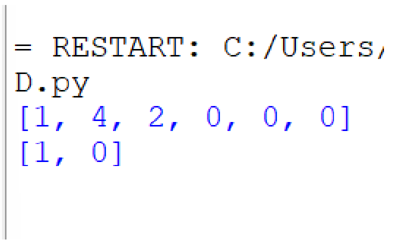
This involves applying the given operations to transform the array and then shifting all zeros to the end.

python



```python
def apply_operations(nums):
    n = len(nums)
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0

    result = [num for num in nums if num != 0] + [0] * nums.count(0)
    return result

# Example usage:
nums1 = [1,2,2,1,1,0]
nums2 = [0,1]
print(apply_operations(nums1))  # Output: [1, 4, 2, 0, 0, 0]
print(apply_operations(nums2))  # Output: [1, 0]
```

Output:



```
= RESTART: C:/Users,
D.py
[1, 4, 2, 0, 0, 0]
[1, 0]
```

These solutions should be efficient enough given the constraints and will handle the specified inputs appropriately.

08.("Average Value of Even Numbers That Are Divisible by Three"), we need to find the average of even numbers in the array that are divisible by 3. Here's how we can implement this:

Iterate through the array to find all even numbers that are divisible by 3.

Calculate the sum of these numbers.

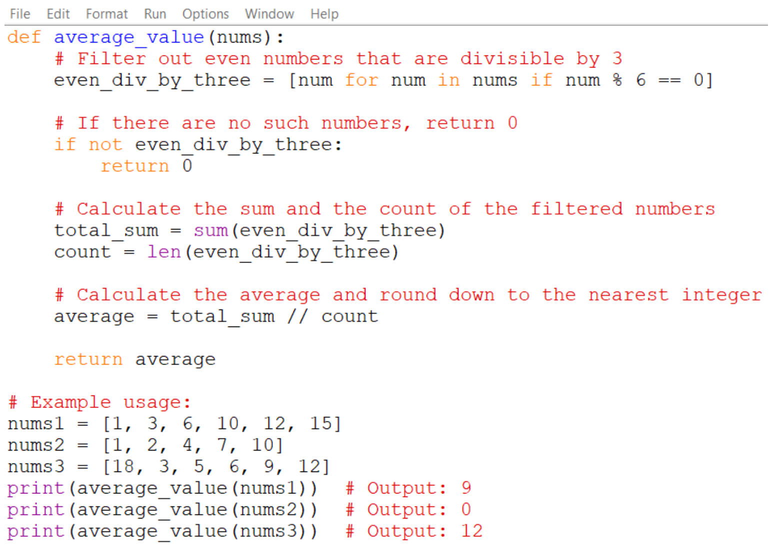
Calculate the count of these numbers.

Compute the average by dividing the sum by the count and rounding down to the nearest integer.

Here's the Python code for this solution:

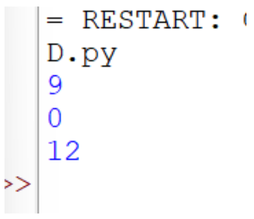
python

Copy code



```python
def average_value(nums):
    # Filter out even numbers that are divisible by 3
    even_div_by_three = [num for num in nums if num % 6 == 0]

    # If there are no such numbers, return 0
    if not even_div_by_three:
        return 0

    # Calculate the sum and the count of the filtered numbers
    total_sum = sum(even_div_by_three)
    count = len(even_div_by_three)

    # Calculate the average and round down to the nearest integer
    average = total_sum // count

    return average

# Example usage:
nums1 = [1, 3, 6, 10, 12, 15]
nums2 = [1, 2, 4, 7, 10]
nums3 = [18, 3, 5, 6, 9, 12]
print(average_value(nums1))  # Output: 9
print(average_value(nums2))  # Output: 0
print(average_value(nums3))  # Output: 12
```

Output;



```
= RESTART: (
D.py
9
0
12
>>
```

Explanation

Filter the numbers: We use a list comprehension to filter out numbers that are both even and divisible by 3. This is equivalent to checking if a number is divisible by 6.

Check for an empty list: If there are no such numbers, return 0.

Calculate sum and count: Compute the sum and the count of the filtered numbers.

Compute the average: Calculate the average by integer division of the total sum by the count. This automatically rounds down to the nearest integer.

This solution ensures we correctly identify the numbers that are both even and divisible by 3, and then calculate their average efficiently.