Odd String Difference

In [1]:
```
def calculate_difference_array(word):
    """Calculate the difference array for a given word."""
    return [ord(word[i + 1]) - ord(word[i]) for i in range(len(word) - 1)]

def find_odd_string(words):
    """Find the odd string out in the list of words."""
    difference_arrays = [calculate_difference_array(word) for word in words]

    for i, diff in enumerate(difference_arrays):
        if difference_arrays.count(diff) == 1:
            return words[i]

    return None

# Example usage
words = ["abc", "bcd", "cde", "xyz", "pqr"]
odd_string = find_odd_string(words)
print(f"The odd string is: {odd_string}")
```

```
The odd string is: None
```

Words Within Two Edits of Dictionary

In [3]:
```
def count_differences(word1, word2):
    """Count the number of different characters between two words."""
    return sum(1 for a, b in zip(word1, word2) if a != b)

def find_similar_words(queries, dictionary):
    """Find words from queries that can match a dictionary word with at most 2
    result = []

    for query in queries:
        for dict_word in dictionary:
            if count_differences(query, dict_word) <= 2:
                result.append(query)
                break
    return result

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

queries = ["yes"]
dictionary = ["not"]
print(find_similar_words(queries, dictionary))
```

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

Destroy Sequential Target

In [5]:
```
from collections import defaultdict

def destroy_sequential_targets(nums, space):

    remainder_count = defaultdict(int)


    for num in nums:
        remainder = num % space
        remainder_count[remainder] += 1


    max_count = 0
    min_value = float('inf')
    for num in nums:
        remainder = num % space
        if remainder_count[remainder] > max_count or (remainder_count[remainde
            max_count = remainder_count[remainder]
            min_value = num

    return min_value


nums1 = [3, 7, 8, 1, 1, 5]
space1 = 2
print(destroy_sequential_targets(nums1, space1))

nums2 = [1, 3, 5, 2, 4, 6]
space2 = 2
print(destroy_sequential_targets(nums2, space2))

nums3 = [6, 2, 5]
space3 = 100
print(destroy_sequential_targets(nums3, space3))
```

```
1
1
2
```

Next Greater Element IV

In [6]:
```
def second_greater(nums):
    n = len(nums)
    result = [-1] * n
    first_stack = []
    second_stack = []

    for i in range(n - 1, -1, -1):
        while second_stack and nums[i] >= second_stack[-1][0]:
            second_stack.pop()

        if second_stack:
            result[i] = second_stack[-1][0]

        while first_stack and nums[i] >= first_stack[-1][0]:
            second_stack.append(first_stack.pop())

        first_stack.append((nums[i], i))

    return result


nums1 = [2, 4, 0, 9, 6]
print(second_greater(nums1))

nums2 = [3, 3]
print(second_greater(nums2))
```

```
[6, 6, 6, -1, -1]
[-1, -1]
```

Average Value of Even Numbers That Are Divisible by Three

In [7]:
```
def average_value_of_even_divisible_by_three(nums):
    filtered_nums = [num for num in nums if num % 6 == 0]
    if not filtered_nums:
        return 0
    return sum(filtered_nums) // len(filtered_nums)

nums1 = [1, 3, 6, 10, 12, 15]
print(average_value_of_even_divisible_by_three(nums1))

nums2 = [1, 2, 4, 7, 10]
print(average_value_of_even_divisible_by_three(nums2))
```

```
9
0
```

Most Popular Video creator

In [9]:
```
from collections import defaultdict

def most_popular_creator(creators, ids, views):
    creator_views = defaultdict(int)
    creator_top_video = {}

    for creator, video_id, view in zip(creators, ids, views):
        creator_views[creator] += view

        if creator not in creator_top_video:
            creator_top_video[creator] = (view, video_id)
        else:
            max_view, max_id = creator_top_video[creator]
            if view > max_view or (view == max_view and video_id < max_id):
                creator_top_video[creator] = (view, video_id)

    max_popularity = max(creator_views.values())


    result = []
    for creator, total_views in creator_views.items():
        if total_views == max_popularity:
            result.append([creator, creator_top_video[creator][1]])

    return result
creators1 = ["alice", "bob", "alice", "chris"]
ids1 = ["one", "two", "three", "four"]
views1 = [5, 10, 5, 4]
print(most_popular_creator(creators1, ids1, views1))

creators2 = ["alice", "alice", "alice"]
ids2 = ["a", "b", "c"]
views2 = [1, 2, 2]
print(most_popular_creator(creators2, ids2, views2))
```

```
[['alice', 'one'], ['bob', 'two']]
[['alice', 'b']]
```

Minimum Addition to Make Integer Beautiful

In [10]:
```
def digit_sum(n):
    return sum(int(digit) for digit in str(n))

def make_n_beautiful(n, target):
    if digit_sum(n) <= target:
        return 0

    x = 0
    power_of_ten = 1

    while digit_sum(n + x) > target:
        remainder = n % (10 * power_of_ten)
        addition = (10 * power_of_ten) - remainder
        x += addition
        power_of_ten *= 10

    return x


n1, target1 = 16, 6
print(make_n_beautiful(n1, target1))

n2, target2 = 467, 6
print(make_n_beautiful(n2, target2))

n3, target3 = 1, 1
print(make_n_beautiful(n3, target3))
```

```
4
109635
0
```

Split Message Based on limit

In [11]:
```
def split_message(message, limit):
    def suffix_length(num_parts):
        return len(f"<{num_parts}/{num_parts}>")

    n = len(message)

    for b in range(1, n + 1):
        suffix_len = suffix_length(b)
        part_len = limit - suffix_len
        if part_len <= 0:
            continue
        if b * part_len >= n:
            break
    else:
        return []

    result = []
    i = 0
    for a in range(1, b + 1):
        suffix = f"<{a}/{b}>"
        part_len = limit - len(suffix)
        result.append(message[i:i + part_len] + suffix)
        i += part_len

    return result

message1 = "this is really a very awesome message"
limit1 = 9
print(split_message(message1, limit1))


message2 = "short message"
limit2 = 15
print(split_message(message2, limit2))
```

```
['thi<1/19>', 's i<2/19>', 's r<3/19>', 'eal<4/19>', 'ly <5/19>', 'a v<6/19
>', 'ery<7/19>', ' aw<8/19>', 'eso<9/19>', 'me<10/19>', ' m<11/19>', 'es<12/1
9>', 'sa<13/19>', 'ge<14/19>', '<15/19>', '<16/19>', '<17/19>', '<18/19>', '<
19/19>']
['short mess<1/2>', 'age<2/2>']
```

In [ ]: