**Python:**

1] You are given an unsorted array of integers. Your task is to find the length of the longest consecutive elements sequence in the array.

Input:

An unsorted array of integers, nums, where the length of nums is at most 10^5. The integers in nums may be duplicates.

Output:

An integer representing the length of the longest consecutive elements sequence in the array.

Explanation:

You need to determine the length of the longest sequence of consecutive elements in the given array.

def longest\_consecutive(nums):

if not nums:

return 0

nums\_set = set(nums)

longest\_sequence = 0

for num in nums\_set:

if num - 1 not in nums\_set:

current\_num = num

current\_sequence = 1

while current\_num + 1 in nums\_set:

current\_num += 1

current\_sequence += 1

longest\_sequence = max(longest\_sequence, current\_sequence)

return longest\_sequence

# Example usage:

nums = [100, 4, 200, 1, 3, 2, 2]

print(longest\_consecutive(nums)) # Output: 4

2]

You are given a grid of size n x 3, where n is a positive integer. Your task is to paint each cell of the grid with exactly one of the three colors: Red, Yellow, or Green. However, you must ensure that no two adjacent cells in the grid have the same color (i.e., no two cells that share vertical or horizontal sides have the same color). You need to return the number of ways you can paint this grid.

Input:

An integer n representing the number of rows in the grid. (1 <= n <= 10^5)

Output:

An integer representing the number of ways you can paint the grid satisfying the given conditions.

Explanation:

You need to determine the number of ways to paint the grid of size n x 3 such that no two adjacent cells have the same color.

# Function to count the number

# of ways to paint N \* 3 grid

# based on given conditions

def waysToPaint(n):

# Count of ways to paint a

# row with same colored ends

same = 6

# Count of ways to paint a row

# with different colored ends

diff = 6

# Traverse up to (N - 1)th row

for \_ in range(n - 1):

# Calculate the count of ways

# to paint the current row

# For same colored ends

sameTmp = 3 \* same + 2 \* diff

# For different colored ends

diffTmp = 2 \* same + 2 \* diff

same = sameTmp

diff = diffTmp

# Print the total number of ways

print(same + diff)

N = int(input("Please enter the number: "))

waysToPaint(N)

input = 5

output = 5118

3]

You are given a string s and a string t. Your task is to find the minimum window in s that contains all the characters in t. The minimum window should be of complexity O(n).

Input:

Two strings s and t where the length of s and t is at most 10^5.

Output:

A string representing the minimum window in s that contains all the characters in t. If there is no such window in s that covers all characters in t, return an empty string.

Explanation:

You need to find the smallest window in s that contains all characters in t in any order.

from collections import Counter

def min\_window(s, t):

if len(s) < len(t):

return ""

# Count occurrences of characters in t

t\_counter = Counter(t)

# Initialize pointers and variables

left = 0

min\_len = float('inf')

min\_window\_start = 0

matched = 0

window\_counts = Counter()

# Iterate over the string

for right in range(len(s)):

# Update window\_counts with the current character

char\_right = s[right]

window\_counts[char\_right] += 1

# Check if the current character is part of t and if its count in window\_counts matches its count in t\_counter

if char\_right in t\_counter and window\_counts[char\_right] == t\_counter[char\_right]:

matched += 1

# Shrink the window from the left if all characters in t are found in the current window

while matched == len(t\_counter):

# Update the minimum window length and its start index if needed

if right - left + 1 < min\_len:

min\_len = right - left + 1

min\_window\_start = left

# Update window\_counts and matched count

char\_left = s[left]

window\_counts[char\_left] -= 1

if char\_left in t\_counter and window\_counts[char\_left] < t\_counter[char\_left]:

matched -= 1

# Move the left pointer to shrink the window

left += 1

# If no minimum window found, return an empty string, otherwise return the minimum window

return "" if min\_len == float('inf') else s[min\_window\_start:min\_window\_start + min\_len]

# Example usage:

s = "ADOBBECODEBANC"

t = "ABCB"

print(min\_window(s, t)) # Output: "ADOBBEC"

4]

You are given a linked list. Your task is to reverse the nodes of the linked list k at a time and return the modified list. The value of k is a positive integer and is less than or equal to the length of the linked list.

Input:

A linked list representing the original list.

Output:

A modified linked list where the nodes are reversed in groups of size k.

Explanation:

You need to reverse the nodes of the linked list k at a time

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def reverse\_k\_group(head, k):

if not head or k == 1:

return head

# Create a dummy node to handle edge cases

dummy = ListNode(0)

dummy.next = head

prev = dummy

while prev:

prev = reverse\_next\_k(prev, k)

return dummy.next

def reverse\_next\_k(prev, k):

# Check if there are at least k nodes remaining to be reversed

end = prev

for \_ in range(k):

end = end.next

if not end:

return None

# Initialize pointers

start = prev.next

curr = start.next

# Reverse k nodes

for \_ in range(k - 1):

start.next = curr.next

curr.next = prev.next

prev.next = curr

curr = start.next

return start

# Helper function to print linked list

def print\_linked\_list(head):

while head:

print(head.val, end=" -> ")

head = head.next

print("None")

# Example usage:

# Create a linked list: 1 -> 2 -> 3 -> 4 -> 5

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

head.next.next.next = ListNode(4)

head.next.next.next.next = ListNode(5)

print("Original linked list:")

print\_linked\_list(head)

k = 2

modified\_head = reverse\_k\_group(head, k)

print("\nModified linked list with nodes reversed in groups of", k, "at a time:")

print\_linked\_list(modified\_head)

**Input:**

Original linked list:

1 -> 2 -> 3 -> 4 -> 5 -> None

**Output:**

Modified linked list with nodes reversed in groups of 2 at a time:

2 -> 1 -> 4 -> 3 -> 5 -> None

**SQL:**

1] Retrieve the Customer IDs of customers who have placed orders in both the East and West regions:

SELECT CustomerID

FROM Orders

WHERE Region IN ('East', 'West')

GROUP BY CustomerID

HAVING COUNT(DISTINCT Region) = 2;

2] Retrieve the Customer Names and their total order amounts for customers who have placed orders in all regions:

SELECT c.CustomerName, SUM(o.OrderAmount) AS TotalOrderAmount

FROM Customers c

JOIN Orders o ON c.CustomerID = o.CustomerID

GROUP BY c.CustomerID, c.CustomerName

HAVING COUNT(DISTINCT o.Region) = (SELECT COUNT(DISTINCT Region) FROM Orders);

3] Retrieve the Customer IDs and the number of orders they've placed where the order amount is greater than the average order amount of their respective regions:

SELECT o.CustomerID, COUNT(\*) AS NumOrders

FROM Orders o

JOIN (

SELECT Region, AVG(OrderAmount) AS AvgOrderAmount

FROM Orders

GROUP BY Region

) AS region\_avg ON o.Region = region\_avg.Region

WHERE o.OrderAmount > region\_avg.AvgOrderAmount

GROUP BY o.CustomerID;