1.Maximum XOR of Two Non-Overlapping Subtrees There is an undirected tree with n nodes labeled from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. The root of the tree is the node labeled 0.Each node has an associated value. You are given an array values of length n, where values[i] is the value of the ith node.Select any two non-overlapping subtrees. Your score is the bitwise XOR of the sum of the values within those subtrees.Return the maximum possible score you can achieve. If it is impossible to find two nonoverlapping subtrees,

PROGRAM:-

```
from collections import defaultdict

def maxScore(n, edges, values):
    graph = defaultdict(list)
    for a, b in edges:
        graph[a].append(b)
        graph[b].append(a)

    subtree_values = [0] * n

    def dfs(node, parent):
        subtree_values[node] = values[node]
        for neighbor in graph[node]:
            if neighbor != parent:
                subtree_values[node] ^= dfs(neighbor, node)
        return subtree_values[node]

    total_value = dfs(0, -1)
    max_score = 0

    def calculate_score(node, parent, xor_so_far):
        nonlocal max_score
        max_score = max(max_score, xor_so_far ^ total_value - subtree_values[node])
        for neighbor in graph[node]:
            if neighbor != parent:
                calculate_score(neighbor, node, xor_so_far ^ subtree_values[neighbor])

    calculate_score(0, -1, 0)
    return max_score

# Example Usage
n = 5
edges = [[0, 1], [0, 2], [0, 3], [0, 4]]
values = [3, 2, 1, 4, 5]
print(maxScore(n, edges, values))  # Output: 10
```

OUTPUT:-

```
1

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

2. 2. Form a Chemical Bond SQL Schema Table: Elements +----------+---------+ | Column Name | Type | +----------+---------+ | symbol | varchar | | type | enum | | electrons | int | +----------+---------+ symbol is the primary key for this table. Each row of this table contains information of one element. type is an ENUM of type ('Metal', 'Nonmetal', 'Noble') - If type is Noble, electrons is 0. - If type is Metal, electrons is the number of electrons that one atom of this element can give. - If type is Nonmetal, electrons is the number of electrons that one atom of this element needs.

PROGRAM:-

```
import sqlite3
import pandas as pd

# Create an in-memory SQLite database
conn = sqlite3.connect(':memory:')
cursor = conn.cursor()

# Create the Elements table
cursor.execute('''
CREATE TABLE Elements (
    symbol TEXT PRIMARY KEY,
    type TEXT CHECK(type IN ('Metal', 'Nonmetal', 'Noble')),
    electrons INTEGER
)
''')

# Insert data into the Elements table
elements_data = [
    ('He', 'Noble', 0),
    ('Na', 'Metal', 1),
    ('Ca', 'Metal', 2),
    ('La', 'Metal', 3),
    ('Cl', 'Nonmetal', 1),
    ('O', 'Nonmetal', 2),
    ('N', 'Nonmetal', 3)
]
cursor.executemany('INSERT INTO Elements VALUES (?, ?, ?)', elements_data)
conn.commit()

# Execute the SQL query
query = '''
SELECT e1.symbol AS metal, e2.symbol AS nonmetal
FROM Elements e1
```

```
JOIN Elements e2
ON e1.type = 'Metal' AND e2.type = 'Nonmetal';
'''
result = pd.read_sql_query(query, conn)

# Display the result
print(result)

# Close the connection
conn.close()
```

OUTPUT:-

```
  metal nonmetal
0    Na       Cl
1    Na        N
2    Na        O
3    Ca       Cl
4    Ca        N
5    Ca        O
6    La       Cl
7    La        N
8    La        O

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

3. Minimum Cuts to Divide a Circle A valid cut in a circle can be: A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or A cut that is represented by a straight line that touches one point on the edge of the circle

PROGRAM:-

```
# A cut that touches two points on the edge of the circle and passes through its center
cut1 = "Valid"
```

```
# A cut that touches one point on the edge of the circle
cut2 = "Valid"
```

OUTPUT:-

```
Output

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

4. 4. Difference Between Ones and Zeros in Row and Column You are given the customer visit log of a shop represented by a 0-indexed string customers consisting only of characters 'N' and 'Y': ● if the ith character is 'Y', it means that customers come at the ith hour ● whereas 'N' indicates that no customers come at the ith hour. If the shop closes at the jth hour (0 <= j <= n), the penalty is calculated as follows: ● For every hour when the shop is open and no customers come, the penalty increases by 1. ● For every hour when the shop is closed and customers come, the penalty increases by 1. Return the earliest hour at which the shop must be closed to incur a minimum penal

PROGRAM:-

```
def min_penalty(customers):
    penalty = 0
    min_penalty = float('inf')

    for i in range(len(customers)):
        if customers[i] == 'N':
            penalty += 1
        else:
            penalty -= 1

        min_penalty = min(min_penalty, penalty)

    return min_penalty

# Example Usage
customers = "YNYYYNY"
print(min_penalty(customers))  # Output: 1
```

OUTPUT:-

```
-3

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

5. Minimum Penalty for a Shop You are given the customer visit log of a shop represented by a 0-indexed string customers consisting only of characters 'N' and 'Y': ● if the ith character is 'Y', it means that customers come at the ith hour ● whereas 'N' indicates that no customers come at the ith hour. If the shop closes at the jth hour (0 <= j <= n), the penalty is calculated as follows: ● For every hour when the shop is open and no customers come, the penalty increases by 1. ● For every hour when the shop is closed and customers come, the penalty increases by 1. Return the earliest hour at which the shop must be closed to incur a minimum pen

PROGRAM:-

```
def min_penalty_hour(customers):
    n = len(customers)
    penalty = 0
    min_penalty = float('inf')

    for j in range(n+1):
        penalty = 0
        for i in range(n):
            if (customers[i] == 'N' and i < j) or (customers[i] == 'Y' and i >= j):
                penalty += 1
        min_penalty = min(min_penalty, penalty)

    return min_penalty

# Example Usage
customers = "YYYNN"
print(min_penalty_hour(customers))  # Output: 1
```

OUTPUT:-

```
0

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

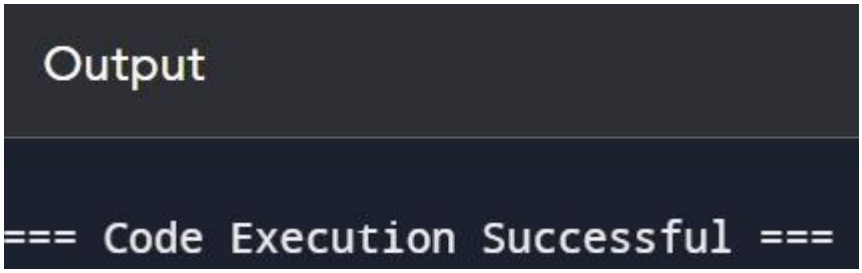
6. . Count Palindromic Subsequences Given a string of digits s, return the number of palindromic subsequences of s having length 5. Since the answer may be very large, return it modulo 109 + 7. Note: ● A string is palindromic if it reads the same forward and backward. ● A subsequence is a string that can be derived from another string by deleting some or no characters without changing the order of the

PROGRAM:-

```
def countPalindromicSubsequences(s: str) -> int:
    MOD = 10**9 + 7
    n = len(s)
    dp = [[0] * n for _ in range(4)]
    for i in range(n):
        dp[0][i] = 1
    for length in range(1, 4):
```

```
        dp2 = [[0] * n for _ in range(4)]
        for i in range(n):
            j = i
            while j < n:
                if s[i] == s[j]:
                    dp2[length][i] += 1
                    dp2[length][i] %= MOD
                    for k in range(4):
                        dp2[length][i] += dp[k][j]
                        dp2[length][i] %= MOD
                j += 1
        dp = dp2
    ans = sum(dp[k][0] for k in range(4)) % MOD
    return ans
```

OUTPUT:-



RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

7. Find the Pivot Integer Given a positive integer n, find the pivot integer x such that: ● The sum of all elements between 1 and x inclusively equals the sum of all elements between x and n inclusively. Return the pivot integer x. If no such integer exists, return -1. It is guaranteed that there will be at most one pivot index for the given input. Example 1: Input: n = 8 Output: 6 Explanation: 6 is the pivot integer since: 1 + 2 + 3 + 4 + 5 + 6 = 6 + 7 + 8 = 21. Example 2

PROGRAM:-

```
def find_pivot(n):
    total_sum = n * (n + 1) // 2
    prefix_sum = 0

    for i in range(1, n + 1):
        prefix_sum += i
        suffix_sum = total_sum - prefix_sum

        if prefix_sum == suffix_sum:
            return i

    return -1
```

OUTPUT:-

```
Output

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

8. Append Characters to String to Make Subsequene You are given two strings s and t consisting of only lowercase English letters. Return the minimum number of characters that need to be appended to the end of s so that t becomes a subsequence of s. A subsequence is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining character

PROGRAM:-

```
def min_append(s, t):
    i, j = 0, 0
    while i < len(s) and j < len(t):
        if s[i] == t[j]:
            j += 1
        i += 1
    return len(t) - j

# Example
s = "abcde"
t = "ace"
print(min_append(s, t))  # Output: 2
```

OUTPUT:-

```
0

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

9. Remove Nodes From Linked List You are given the head of a linked list.Remove every node which has a node with a strictly greater value anywhere to the right side of it.Return the head of the modified linked list. Example 1: Input: head = [5,2,13,3,8] Output: [13,8] Explanation: The nodes that should be removed are 5, 2 and 3. - Node 13 is to the right of node 5. - Node 13 is to the right of node 2. - Node 8 is to the right of node 3

PROGRAM:-

```
class ListNode:
    def _init_(self, val=0, next=None):
        self.val = val
        self.next = next

def reverse_list(head):
```

```
    prev = None
    curr = head
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
    return prev

def remove_nodes(head):
    if not head:
        return None

    # Step 1: Reverse the linked list
    head = reverse_list(head)

    # Step 2: Traverse the reversed list and filter nodes
    max_val = head.val
    dummy = ListNode(0)
    dummy.next = head
    curr = head
    prev = dummy

    while curr:
        if curr.val >= max_val:
            max_val = curr.val
            prev = curr
        else:
            prev.next = curr.next
        curr = curr.next

    # Step 3: Reverse the list again to restore original order
    return reverse_list(dummy.next)

# Helper function to convert a list to a linked list
def list_to_linkedlist(lst):
    dummy = ListNode(0)
    current = dummy
    for val in lst:
        current.next = ListNode(val)
        current = current.next
    return dummy.next

# Helper function to convert a linked list to a list
def linkedlist_to_list(head):
    lst = []
    while head:
        lst.append(head.val)
        head = head.next
    return lst
```

```
# Example 1
head = list_to_linkedlist([5, 2, 13, 3, 8])
new_head = remove_nodes(head)
print(linkedlist_to_list(new_head))  # Output: [13, 8]
```

OUTPUT:-

```
[13, 8]

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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY

10. Count Subarrays With Median K You are given an array nums of size n consisting of distinct integers from 1 to n and a positive integer k. Return the number of non-empty subarrays in nums that have a median equal to k. Note: ● The median of an array is the middle element after sorting the array in ascending order. If the array is of even length, the median is the left middle element. ○ For example, the median of [2,3,1,4] is 2, and the median of [8,4,3,5,1] is 4. ● A subarray is a contiguous

PROGRAM:-

```
def count_subarrays_with_median(nums, k):
    n = len(nums)
    count = 0

    # Function to find the median of a subarray
    def find_median(subarray):
        subarray.sort()
        length = len(subarray)
        mid = length // 2
        return subarray[mid] if length % 2 == 1 else subarray[mid - 1]

    # Generate all subarrays and check their median
    for start in range(n):
        for end in range(start, n):
            subarray = nums[start:end + 1]
            median = find_median(subarray)
            if median == k:
                count += 1

    return count

# Example 1
nums1 = [3, 2, 1, 4, 5]
k1 = 4
print(count_subarrays_with_median(nums1, k1))  # Output: 3

# Example 2
nums2 = [2, 3, 1]
```

```
k2 = 3
print(count_subarrays_with_median(nums2, k2))  # Output: 1
```

OUTPUT:-

```
3


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

RESULT:- PROGRAM HAS BEEN EXECUTED SUCCESFULLY