**ASSIGNMENT [11.06.2024]**

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, return 0. Note that: ● The subtree of a node is the tree consisting of that node and all of its descendants. ● Two subtrees are non-overlapping if they do not share any common node**

from collections import defaultdict, deque

class TrieNode:

def \_\_init\_\_(self):

self.children = {}

self.value = None

class Trie:

def \_\_init\_\_(self):

self.root = TrieNode()

def insert(self, num):

node = self.root

for i in range(31, -1, -1):

bit = (num >> i) & 1

if bit not in node.children:

node.children[bit] = TrieNode()

node = node.children[bit]

node.value = num

def find\_max\_xor(self, num):

node = self.root

max\_xor = 0

for i in range(31, -1, -1):

bit = (num >> i) & 1

toggled\_bit = 1 - bit

if toggled\_bit in node.children:

max\_xor = (max\_xor << 1) | 1

node = node.children[toggled\_bit]

else:

max\_xor = (max\_xor << 1) | 0

node = node.children[bit]

return max\_xor

def dfs(node, parent, adj, values, subtree\_sum):

subtree\_sum[node] = values[node]

for neighbor in adj[node]:

if neighbor != parent:

dfs(neighbor, node, adj, values, subtree\_sum)

subtree\_sum[node] += subtree\_sum[neighbor]

def maximum\_xor\_of\_two\_subtrees(n, edges, values):

# Build the adjacency list

adj = defaultdict(list)

for u, v in edges:

adj[u].append(v)

adj[v].append(u)

# Step 1: Calculate subtree sums

subtree\_sum = [0] \* n

dfs(0, -1, adj, values, subtree\_sum)

# Step 2: Find the maximum XOR of two non-overlapping subtrees

trie = Trie()

max\_xor = 0

def dfs\_max\_xor(node, parent):

nonlocal max\_xor

current\_sum = subtree\_sum[node]

if trie.root.children:

max\_xor = max(max\_xor, trie.find\_max\_xor(current\_sum))

trie.insert(current\_sum)

for neighbor in adj[node]:

if neighbor != parent:

dfs\_max\_xor(neighbor, node)

trie.insert(current\_sum)

dfs\_max\_xor(0, -1)

return max\_xor

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. Two elements can form a bond if one of them is 'Metal' and the other is 'Nonmetal'.Write an SQL query to find all the pairs of elements that can form a bond.Return the result table in any order.The query result format is in the**

SELECT

e1.symbol AS element1,

e2.symbol AS element2

FROM

Elements e1

JOIN

Elements e2

ON

(e1.type = 'Metal' AND e2.type = 'Nonmetal')

OR (e1.type = 'Nonmetal' AND e2.type = 'Metal');

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 and its center. Some valid and invalid cuts are shown in the figures below. Given the integer n, return the minimum number of cuts needed to divide a circle into n equal slices.**

def minCutsToDivideCircle(n):

if n == 1:

return 0

elif n % 2 == 0:

return n // 2

else:

return n

# Example usage:

print(minCutsToDivideCircle(1)) # Output: 0

print(minCutsToDivideCircle(4)) # Output: 2

print(minCutsToDivideCircle(3)) # Output: 3

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 penalty. Note that if a shop closes at the jth hour, it means the shop is closed at the hour j. Example 1: Input: customers = "YYNY" Output: 2 Explanation: - Closing the shop at the 0th hour incurs in 1+1+0+1 = 3 penalty. - Closing the shop at the 1st hour incurs in 0+1+0+1 = 2 penalty. - Closing the shop at the 2nd hour incurs in 0+0+0+1 = 1 penalty. - Closing the shop at the 3rd hour incurs in 0+0+1+1 = 2 penalty. - Closing the shop at the 4th hour incurs in 0+0+1+0 = 1 penalty. Closing the shop at 2nd or 4th hour gives a minimum penalty. Since 2 is earlier, the optimal closing time is 2.**def min\_penalty\_closing\_time(customers):

n = len(customers)

penalty\_open = [0] \* (n + 1)

penalty\_close = [0] \* (n + 1)

# Calculate the penalty for open hours

for i in range(n):

penalty\_open[i + 1] = penalty\_open[i] + (1 if customers[i] == 'N' else 0)

# Calculate the penalty for closed hours

for i in range(n - 1, -1, -1):

penalty\_close[i] = penalty\_close[i + 1] + (1 if customers[i] == 'Y' else 0)

# Find the hour with the minimum penalty

min\_penalty = float('inf')

min\_hour = 0

for i in range(n + 1):

total\_penalty = penalty\_open[i] + penalty\_close[i]

if total\_penalty < min\_penalty:

min\_penalty = total\_penalty

min\_hour = i

return min\_hour

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 penalty. Note that if a shop closes at the jth hour, it means the shop is closed at**

def min\_penalty\_hour(customers):

n = len(customers)

min\_penalty = float('inf')

min\_hour = 0

for hour in range(n + 1):

open\_penalty = 0

closed\_penalty = 0

# Calculate the penalty for open hours

for i in range(hour):

if customers[i] == 'N':

open\_penalty += 1

# Calculate the penalty for closed hours

for i in range(hour, n):

if customers[i] == 'Y':

closed\_penalty += 1

# Total penalty for closing the shop at this hour

total\_penalty = open\_penalty + closed\_penalty

# Update the minimum penalty and hour

if total\_penalty < min\_penalty:

min\_penalty = total\_penalty

min\_hour = hour

return min\_hour

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 remaining character**

MOD = 10\*\*9 + 7

def count\_palindromic\_subsequences(s):

n = len(s)

dp = [[0] \* n for \_ in range(n)]

# Initialize base cases for subsequences of length 1

for i in range(n):

dp[i][i] = 1

# Calculate subsequences of length 2 to 5

for length in range(2, 6):

for i in range(n - length + 1):

j = i + length - 1

if length == 2:

dp[i][j] = 2 if s[i] == s[j] else 1

else:

dp[i][j] = (2 \* dp[i + 1][j - 1] + 2) % MOD

for k in range(i + 1, j):

if s[i] == s[k]:

dp[i][j] = (dp[i][j] + dp[i + 1][k - 1] + 1) % MOD

if s[j] == s[k]:

dp[i][j] = (dp[i][j] + dp[k + 1][j - 1] + 1) % MOD

# Sum all palindromic subsequences of length 5

count = sum(dp[i][i + 4] for i in range(n - 4)) % MOD

return count

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**

def find\_pivot\_integer(n):

total\_sum = (n \* (n + 1)) // 2

left\_sum = 0

for x in range(1, n + 1):

left\_sum += x

right\_sum = total\_sum - left\_sum

if left\_sum == right\_sum:

return x

return -1

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 characters.**

def min\_characters\_to\_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 usage:

s = "abcde"

t = "ace"

print(min\_characters\_to\_append(s, t)) # Output: 2 (append "d" and "e" to make "ace" subsequence of "abcde")

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**

class ListNode:

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

self.val = val

self.next = next

def remove\_nodes(head):

dummy = ListNode(0)

dummy.next = head

last\_survivor = dummy

while head and head.next:

if head.val < head.next.val:

head = head.next

else:

last\_survivor.next = head.next

head = head.next

return dummy.next

# Helper function to print the linked list

def print\_linked\_list(head):

while head:

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

head = head.next

print("None")

# Example usage:

head = ListNode(5)

head.next = ListNode(2)

head.next.next = ListNode(13)

head.next.next.next = ListNode(3)

head.next.next.next.next = ListNode(8)

print("Original linked list:")

print\_linked\_list(head)

head = remove\_nodes(head)

print("\nModified linked list after removal:")

print\_linked\_list(head)

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 part**

def count\_subarrays\_with\_median\_k(nums, k):

count = 0

left = 0

for right in range(len(nums)):

# Calculate the size of the current window

window\_size = right - left + 1

# If the size of the window is odd and k is the middle element

if window\_size % 2 != 0 and nums[left + window\_size // 2] == k:

count += 1

# If the size of the window is even and either k or k-1 is the left middle element

elif window\_size % 2 == 0 and nums[left + window\_size // 2] in (k, k - 1):

count += 1

return count

# Example usage:

nums = [2, 3, 1, 4]

k = 2

print(count\_subarrays\_with\_median\_k(nums, k)) # Output: 2 (subarrays [2] and [2, 3, 1, 4])

nums = [8, 4, 3, 5, 1]

k = 4

print(count\_subarrays\_with\_median\_k(nums, k)) # Output: 1 (subarray [4, 3, 5])