1.MaximumXORofTwoNon-OverlappingSubtrees.

from collections import defaultdict

def maximumXORofTwoNonOverlappingSubtrees(n, edges, values):

# Create the adjacency list for the tree

tree = defaultdict(list)

for a, b in edges:

tree[a].append(b)

tree[b].append(a)

subtree\_sum = [0] \* n

def dfs(node, parent):

subtree\_sum[node] = values[node]

for neighbor in tree[node]:

if neighbor != parent:

dfs(neighbor, node)

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

dfs(0, -1)

seen\_sums = set()

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

current\_sum = subtree\_sum[node]

max\_xor = 0

for neighbor in tree[node]:

if neighbor != parent:

max\_xor = max(max\_xor, find\_max\_xor(neighbor, node))

for s in seen\_sums:

max\_xor = max(max\_xor, s ^ current\_sum)

seen\_sums.add(current\_sum)

return max\_xor

result = find\_max\_xor(0, -1)

return result

n = 5

edges = [[0,1], [0,2], [1,3], [1,4]]

values = [1, 2, 3, 4, 5]

print(maximumXORofTwoNonOverlappingSubtrees(n, edges, values))

2.FormaChemicalBond.

import sqlite3

conn = sqlite3.connect(':memory:')

cursor = conn.cursor()

cursor.execute('''CREATE TABLE Elements ( symbol VARCHAR PRIMARY KEY,type ENUM('Metal', 'Nonmetal', 'Noble'), electrons INT ) ''')

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 (symbol, type, electrons) VALUES (?, ?, ?)', elements\_data)

query = '''

SELECT e1.symbol AS metal, e2.symbol AS nonmetal

FROM Elements e1, Elements e2

WHERE e1.type = 'Metal' AND e2.type = 'Nonmetal'

'''

cursor.execute(query)

results = cursor.fetchall()

for row in results:

print(row)

conn.close()

3. Minimum Cuts to Divide a Circle.

def minimumCuts(n):

if n == 1:

return 0

elif n % 2 == 0:

return n // 2

else:

return n

# Example Usage

if \_name\_ == "\_main\_":

test\_cases = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for n in test\_cases:

print(f"Minimum cuts to divide the circle into {n} equal slices: {minimumCuts(n)}")

4. Difference Between Ones and Zeros in Row and Column.

def earliestClosingHour(customers: str) -> int:

n = len(customers)

min\_penalty = float('inf')

min\_hour = 0

for hour in range(n + 1):

penalty = 0

for i in range(n):

if hour == i and customers[i] == 'N':

penalty += 1 # Penalty for closing shop when customers arrive

elif hour != i and customers[i] == 'Y':

penalty += 1 # Penalty for not closing shop when no customers arrive

if penalty < min\_penalty:

min\_penalty = penalty

min\_hour = hour

return min\_hour

print(earliestClosingHour("YYNY"))

print(earliestClosingHour("NNNNN"))

print(earliestClosingHour("YYYY"))

5. Minimum Penalty for a Shop.

def earliestClosingHour(customers: str) -> int:

n = len(customers)

min\_penalty = float('inf')

min\_hour = 0

for hour in range(n + 1):

penalty = 0

for i in range(n):

if hour == i and customers[i] == 'N':

penalty += 1 # Penalty for closing shop when customers arrive

elif hour != i and customers[i] == 'Y':

penalty += 1 # Penalty for not closing shop when no customers arrive

if penalty < min\_penalty:

min\_penalty = penalty

min\_hour = hour

return min\_hour

print(earliestClosingHour("YYN"))

6. Count Palindromic Subsequences.

def countPalindromicSubsequences(s: str) -> int:

MOD = 10\*\*9 + 7

n = len(s)

dp = [[0] \* 10 for \_ in range(n)]

for i in range(n):

dp[i][int(s[i])] = 1

for i in range(1, n):

for j in range(10):

for k in range(10):

if s[i] == str(k):

dp[i][j] += dp[i-1][k]

dp[i][j] %= MOD

return sum(dp[-1]) % MOD

print(countPalindromicSubsequences("103301"))

print(countPalindromicSubsequences("0000000"))

print(countPalindromicSubsequences("9999900000"))

7. Find the Pivot Integer.

def findPivotInteger(n: int) -> int:

total\_sum = (n \* (n + 1)) // 2 # Sum of numbers from 1 to n

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

print(findPivotInteger(8))

print(findPivotInteger(1))

print(findPivotInteger(4))

8. Append Characters to String to Make Subsequene.

def minCharsToAppend(s: str, t: str) -> int:

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

print(minCharsToAppend("coaching", "coding"))

print(minCharsToAppend("abcde", "a"))

print(minCharsToAppend("z", "abcde"))

9. Remove Nodes From Linked List.

class ListNode:

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

self.val = val

self.next = next

def reverseList(head):

prev = None

curr = head

while curr:

next\_node = curr.next

curr.next = prev

prev = curr

curr = next\_node

return prev

def removeNodes(head):

# Step 1: Reverse the linked list

head = reverseList(head)

stack = []

dummy = ListNode(0)

curr = dummy

while head:

if not stack or head.val >= stack[-1]:

stack.append(head.val)

curr.next = ListNode(head.val)

curr = curr.next

head = head.next

return reverseList(dummy.next)

def createLinkedList(arr):

dummy = ListNode(0)

curr = dummy

for val in arr:

curr.next = ListNode(val)

curr = curr.next

return dummy.next

def linkedListToList(head):

result = []

while head:

result.append(head.val)

head = head.next

return result

head = createLinkedList([5, 2, 13, 3, 8])

new\_head = removeNodes(head)

print(linkedListToList(new\_head))

head = createLinkedList([1, 1, 1, 1])

new\_head = removeNodes(head)

print(linkedListToList(new\_head))

10. Count Subarrays With Median K.

def countSubarraysWithMedianK(nums, k):

n = len(nums)

k\_index = nums.index(k)

balance\_count = {0: 1}

balance = 0

count = 0

# Iterate through the array

for i in range(n):

if nums[i] > k:

balance += 1

elif nums[i] < k:

balance -= 1

# When i is to the right of k\_index, we check the balance

if i >= k\_index:

# We need balance or balance - 1 to have a valid subarray

count += balance\_count.get(balance, 0)

count += balance\_count.get(balance - 1, 0)

# Update balance\_count

if i < k\_index:

balance\_count[balance] = balance\_count.get(balance, 0) + 1

return count

nums1 = [3, 2, 1, 4, 5]

k1 = 4

print(countSubarraysWithMedianK(nums1, k1))

nums2 = [2, 3, 1]

k2 = 3

print(countSubarraysWithMedianK(nums2, k2))