**1.Maximum XOR**

from collections import defaultdict

class Tree:

def \_\_init\_\_(self, n, edges, values):

self.n = n

self.edges = edges

self.values = values

self.tree = defaultdict(list)

self.subtree\_sum = [0] \* n

self.construct\_tree()

def construct\_tree(self):

for u, v in self.edges:

self.tree[u].append(v)

self.tree[v].append(u)

def calculate\_subtree\_sums(self, node, parent):

subtree\_sum = self.values[node]

for neighbor in self.tree[node]:

if neighbor != parent:

subtree\_sum += self.calculate\_subtree\_sums(neighbor, node)

self.subtree\_sum[node] = subtree\_sum

return subtree\_sum

def max\_xor\_of\_two\_subtrees(self):

self.calculate\_subtree\_sums(0, -1)

max\_xor = 0

subtree\_sums = set(self.subtree\_sum[1:])

for sum\_value in subtree\_sums:

for other\_sum in subtree\_sums:

if sum\_value != other\_sum:

max\_xor = max(max\_xor, sum\_value ^ other\_sum)

return max\_xor

def max\_xor\_of\_two\_non\_overlapping\_subtrees(n, edges, values):

if n < 2:

return 0

tree = Tree(n, edges, values)

return tree.max\_xor\_of\_two\_subtrees()

n = 6

edges = [[0, 1], [0, 2], [1, 3], [1, 4], [2, 5]]

values = [2, 8, 3, 6, 2, 5]

print(max\_xor\_of\_two\_non\_overlapping\_subtrees(n, edges, values))

**2.Create table**

CREATE TABLE Elements (

symbol VARCHAR PRIMARY KEY,

type ENUM('Metal', 'Nonmetal', 'Noble'),

electrons INT

);

**3.minimum cuts divide circle**

def min\_cuts\_to\_divide\_circle(n):

if n == 1:

return 0

return n if n % 2 == 1 else n // 2

print(min\_cuts\_to\_divide\_circle(4))

**4.Difference between ones and zeros in row and column**

def bestClosingTime(customers: str) -> int:

n = len(customers)

penalty\_open = 0

penalty\_close = customers.count('Y')

min\_penalty = penalty\_open + penalty\_close

best\_hour = 0

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

if customers[i - 1] == 'Y':

penalty\_close -= 1

else:

penalty\_open += 1

current\_penalty = penalty\_open + penalty\_close

if current\_penalty < min\_penalty:

min\_penalty = current\_penalty

best\_hour = i

return best\_hour

print(bestClosingTime("YYNY"))

**5.Minimum penalty for a shop**

def minimum\_penalty(customers):

n = len(customers)

min\_penalty = float('inf')

min\_hour = 0

left\_N = [0] \* (n + 1)

right\_Y = [0] \* (n + 1)

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

left\_N[i] = left\_N[i - 1] + (1 if customers[i - 1] == 'N' else 0)

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

right\_Y[i] = right\_Y[i + 1] + (1 if customers[i] == 'Y' else 0)

for j in range(n + 1):

penalty = left\_N[j] + right\_Y[j]

if penalty < min\_penalty:

min\_penalty = penalty

min\_hour = j

return min\_hour

print(minimum\_penalty("YYNY"))

**6.count palindrome subsequence**

def count\_palindromic\_subsequences(s):

MOD = 10\*\*9 + 7

n = len(s)

if n < 5:

return 0

count = 0

for i in range(n):

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

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

for l in range(k+1, n):

for m in range(l+1, n):

if s[i] == s[m] and s[j] == s[l]:

count = (count + 1) % MOD

return count

print(count\_palindromic\_subsequences("103301"))

**7.Pivot integer**

def find\_pivot\_integer(n):

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

running\_sum = 0

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

running\_sum += x

if running\_sum == total\_sum - running\_sum + x:

return x

return -1

print(find\_pivot\_integer(8))

**8.Append characters**

def append\_characters(s: str, t: str) -> int:

m, n = len(s), len(t)

j = 0

for i in range(m):

if j < n and s[i] == t[j]:

j += 1

return n - j

print(append\_characters("coaching", "coding"))

**9.Remove nodes from linked list**

class ListNode:

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

self.val = val

self.next = next

def remove\_nodes(head: ListNode) -> ListNode:

if not head or not head.next:

return head

prev, current = None, head

while current:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

max\_val = float('-inf')

current = prev

new\_head = None

while current:

if current.val >= max\_val:

max\_val = current.val

if new\_head is None:

new\_head = ListNode(current.val)

new\_tail = new\_head

else:

new\_tail.next = ListNode(current.val)

new\_tail = new\_tail.next

current = current.next

prev, current = None, new\_head

while current:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

return prev

def print\_list(head):

while head:

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

head = head.next

print("None")

head = ListNode(5, ListNode(2, ListNode(13, ListNode(3, ListNode(8)))))

new\_head = remove\_nodes(head)

print\_list(new\_head)

**10.Count subarrays with median k**

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

n = len(nums)

k\_index = nums.index(k)

left\_counts = {0: 1}

balance = 0

count = 0

for i in range(k\_index, -1, -1):

if nums[i] < k:

balance -= 1

elif nums[i] > k:

balance += 1

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

balance = 0

for i in range(k\_index, n):

if nums[i] < k:

balance -= 1

elif nums[i] > k:

balance += 1

count += left\_counts.get(-balance, 0)

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

return count

print(count\_subarrays\_with\_median\_k([3, 2, 1, 4, 5], 4))