1. Convert the Temperature

You are given a non-negative floating point number rounded to two decimal places celsius, that denotes the temperature in Celsius.You should convert Celsius into Kelvin and Fahrenheit and return it as an array

**Code with output:**

def convert(celsius):

kelvin = celsius + 273.15

fahrenheit = celsius \* 1.80 + 32.00

return [round(kelvin, 5), round(fahrenheit, 5)]

print(convert(36.50)) # Output: [309.65000, 97.70000]

print(convert (122.11)) # Output: [395.26000, 251.79800]

1. Number of Subarrays With LCM Equal to K

**Code with output:**

from math import gcd

from functools import reduce

def lcm(a, b):

return a \* b // gcd(a, b)

def l(lst):

return reduce(lcm, lst)

def x(nums, k):

count = 0

n = len(nums)

for i in range(n):

current\_lcm = 1

for j in range(i, n):

current\_lcm = lcm(current\_lcm, nums[j])

if current\_lcm == k:

count += 1

elif current\_lcm > k:

break

return count

print(x([3, 6, 2, 7, 1], 6)) # Output: 4

print(x([3], 2)) # Output: 0

1. Minimum Number of Operations to Sort a Binary Tree by Level

**Code with output:**

from collections import deque

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def m(arr):

n = len(arr)

sorted\_arr = sorted(range(n), key=lambda i: arr[i])

visited = [False] \* n

swaps = 0

for i in range(n):

if visited[i] or sorted\_arr[i] == i:

continue

cycle\_size = 0

x = i

while not visited[x]:

visited[x] = True

x = sorted\_arr[x]

cycle\_size += 1

if cycle\_size > 0:

swaps += cycle\_size - 1

return swaps

def x(root):

if not root:

return 0

queue = deque([root])

s = 0

while queue:

n= len(queue)

l= []

for \_ in range(n):

node = queue.popleft()

l.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

s+= m(l)

return s

root1 = TreeNode(1)

root1.left = TreeNode(4)

root1.right = TreeNode(3)

root1.left.left = TreeNode(7)

root1.left.right = TreeNode(6)

root1.right.left = TreeNode(8)

root1.right.right = TreeNode(5)

root1.right.left.left = TreeNode(9)

root1.right.right.left = TreeNode(10)

print(x(root1)) # Output: 3

1. Maximum Number of Non-overlapping Palindrome Substrings

Code with output:

def m(s, k):

n = len(s)

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

for i in range(n):

dp[i][i] = True

for i in range(n - 1):

if s[i] == s[i + 1]:

dp[i][i + 1] = True

for length in range(3, n + 1):

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

j = i + length - 1

if s[i] == s[j] and dp[i + 1][j - 1]:

dp[i][j] = True

c = [0] \* n

for j in range(n):

for i in range(j + 1):

if j - i + 1 >= k and dp[i][j]:

if i == 0:

c[j] = max(c[j], 1)

else:

c[j] = max(c[j], c[i - 1] + 1)

return max(c)

s1 = "abaccdbbd"

k1 = 3

print(m (s1, k1)) # Output: 2

s2 = "adbcda"

k2 = 2

print(m (s2, k2)) # Output: 0

1. Minimum Cost to Buy Apples

Code with output:

import heapq

from collections import defaultdict, deque

def d(start, n, adj):

dist = [float('inf')] \* n

dist[start] = 0

pq = [(0, start)]

while pq:

d, u = heapq.heappop(pq)

if d > dist[u]:

continue

for v, w in adj[u]:

if dist[u] + w < dist[v]:

dist[v] = dist[u] + w

heapq.heappush(pq, (dist[v], v))

return dist

def m(n, roads, x, k):

adj = defaultdict(list)

for a, b, cost in roads:

adj[a-1].append((b-1, cost))

adj[b-1].append((a-1, cost))

l= []

for i in range(n):

dist = d(i, n, adj)

m= float('inf')

for j in range(n):

if i == j:

cost =x[j]

else:

cost = dist[j] +x[j] + dist[j] \* k

m = min(m, cost)

l.append(m)

return l

n1 = 4

roads1 = [[1, 2, 4], [2, 3, 2], [2, 4, 5], [3, 4, 1], [1, 3, 4]]

x = [56, 42, 102, 301]

k1 = 2

print(m(n1, roads1, x, k1)) # Output: [54, 42, 48, 51]

n2 = 3

roads2 = [[1, 2, 5], [2, 3, 1], [3, 1, 2]]

x = [2, 3, 1]

k2 = 3

print(m (n2, roads2, x, k2)) # Output: [2, 3, 1]

1. Customers With Strictly Increasing Purchases

Code with output:

WITH yearly\_totals AS (

SELECT

customer\_id,

EXTRACT(YEAR FROM order\_date) AS year,

SUM(price) AS yearly\_total

FROM Orders

GROUP BY customer\_id, year

),

yearly\_growth AS (

SELECT

customer\_id,

year,

yearly\_total,

LAG(yearly\_total) OVER (PARTITION BY customer\_id ORDER BY year) AS prev\_yearly\_total

FROM yearly\_totals

)

SELECT DISTINCT customer\_id

FROM yearly\_growth

WHERE prev\_yearly\_total IS NULL

OR yearly\_total > prev\_yearly\_total

GROUP BY customer\_id

HAVING COUNT(\*) = COUNT(prev\_yearly\_total IS NOT NULL AND yearly\_total > prev\_yearly\_total);

7.Number of Unequal Triplets in Array

Code with output:

def c(a):

n = len(a)

count = 0

for i in range(n):

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

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

if a[i] != a[j] and a[i] != a[k] and a[j] != a[k]:

count += 1

return count

a = [4, 4, 2, 4, 3]

print(c(a)) # Output: 3

a = [1, 1, 1, 1, 1]

print(c (a)) # Output: 0

8.Closest Nodes Queries in a Binary Search Tree

Code with output:

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def inorder(root):

if not root:

return []

return inorder (root.left) + [root.val] + inorder (root.right)

def c(root, queries):

sorted\_vals = inorder (root)

result = []

for query in queries:

mini = -1

maxi = -1

for val in sorted\_vals:

if val <= query:

mini = max(mini, val)

if val >= query:

maxi = min(maxi, val) if maxi != -1 else val

result.append([mini, maxi])

return result

root = TreeNode(6, TreeNode(2, TreeNode(1), TreeNode(4)), TreeNode(13, TreeNode(9), TreeNode(15, TreeNode(14))))

queries = [2, 5, 16]

print(c(root, queries)) # Output: [[2, 2], [4, 6], [15, -1]]

9.Minimum Fuel Cost to Report to the Capital

Code with output:

from collections import defaultdict

def b (roads):

adj = defaultdict(list)

for u, v in roads:

adj [u].append(v)

adj [v].append(u)

return adj

def dfs(adj, seats, city, parent):

m= seats[city]

for I in m[city]:

if i != parent:

m = max(m, dfs(adj ,seats, i, city))

return m

def mi(roads, seats):

adj = b(roads)

m = dfs(adj, seats, 0, -1)

return m - 1

roads = [[0,1],[0,2],[0,3]]

seats = [5, 5, 5, 5]

print(mi(roads, seats)) # Output: 3

10. Number of Beautiful Partitions

Code with output:

def c(s, k, m):

MOD = 10\*\*9 + 7

def p(digit):

return digit in {'2', '3', '5', '7'}

n = len(s)

dp = [[0] \* (k + 1) for \_ in range(n + 1)]

dp[0][0] = 1

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

for i in range(1, n - length + 2):

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

if p(s[i - 1]) and not p (s[i + length - 2]):

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

return dp[n][k]

s = "23542185131"

k = 3

m = 2

print(c(s, k, m)) # Output: 3