**Assignment (12-06-2024)**

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 ans = [kelvin, fahrenheit]. Return the array ans. Answers within 10-5 of the actual answer will be accepted

def convert\_temperature(celsius: float) -> list:

kelvin = celsius + 273.15

fahrenheit = celsius \* 1.80 + 32.00

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

celsius = 36.50

ans = convert\_temperature(celsius)

print(ans)

**Output:**

[309.65000, 97.70000]

1. Number of Subarrays With LCM Equal to K

Given an integer array nums and an integer k, return the number of subarrays of nums where the least common multiple of the subarray's elements is k.A subarray is a contiguous non- empty sequence of elements within an array.The least common multiple of an array is the smallest positive integer that is divisible by all the array element

from math import gcd

from functools import reduce

def lcm(x, y):

return x \* y // gcd(x, y)

def count\_subarrays\_with\_lcm(nums, k):

n = len(nums)

count = 0

for i in range(n):

current\_lcm = nums[i]

for j in range(i, n):

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

if current\_lcm == k:

count += 1

if current\_lcm > k:

break

return count

nums = [3, 6, 2, 7, 1]

k = 6

print(count\_subarrays\_with\_lcm(nums, k))

**Output:**

4

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

You are given the root of a binary tree with unique values.In one operation, you can choose any two nodes at the same level and swap their values.Return the minimum number of operations needed to make the values at each level sorted in a strictly increasing

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 minSwapsToSort(arr):

n = len(arr)

ans = 0

temp = arr.copy()

temp.sort()

index\_map = {v: i for i, v in enumerate(arr)}

for i in range(n):

if arr[i] != temp[i]:

ans += 1

swap\_with\_idx = index\_map[temp[i]]

index\_map[arr[i]] = swap\_with\_idx

arr[i], arr[swap\_with\_idx] = arr[swap\_with\_idx], arr[i]

return ans

def minOperationsToSortTreeLevels(root):

if not root:

return 0

level\_map = {}

queue = deque([(root, 0)])

while queue:

node, level = queue.popleft()

if level not in level\_map:

level\_map[level] = []

level\_map[level].append(node.val)

if node.left:

queue.append((node.left, level + 1))

if node.right:

queue.append((node.right, level + 1))

total\_swaps = 0

for level in level\_map:

level\_values = level\_map[level]

total\_swaps += minSwapsToSort(level\_values)

return total\_swaps

root = TreeNode(1)

root.left = TreeNode(4)

root.right = TreeNode(3)

root.left.left = TreeNode(7)

root.left.right = TreeNode(6)

root.right.left = TreeNode(8)

root.right.right = TreeNode(5)

root.left.left.left = TreeNode(9)

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

print(minOperationsToSortTreeLevels(root))

**Output:**

3

1. . Maximum Number of Non-overlapping Palindrome Substrings

You are given a string s and a positive integer k.Select a set of non-overlapping substrings from the string s that satisfy the following conditions: ● The length of each substring is at least k. ● Each substring is a palindrome. Return the maximum number of substrings in an optimal selection.A substring is a contiguous sequence of characters within a string.

def max\_palindrome\_substrings(s: str, k: int) -> int:

n = len(s)

def is\_palindrome(left: int, right: int) -> bool:

while left < right:

if s[left] != s[right]:

return False

left += 1

right -= 1

return True

dp = [0] \* n

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

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

if is\_palindrome(j, i):

if j == 0:

dp[i] = max(dp[i], 1)

else:

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

return max(dp)

s = "abaccdbbd"

k = 3

print(max\_palindrome\_substrings(s, k))

**Output:**

2

1. Minimum Cost to Buy Apples

You are given a positive integer n representing n cities numbered from 1 to n. You are also given a 2D array roads, where roads[i] = [ai, bi, costi] indicates that there is a bidirectional road between cities ai and bi with a cost of traveling equal to costi. You can buy apples in any city you want, but some cities have different costs to buy apples. You are given the array appleCost where appleCost[i] is the cost of buying one apple from city i. You start at some city, traverse through various roads, and eventually buy exactly one apple from any city. After you buy that apple, you have to return back to the city you started at, but now the cost of all the roads will be multiplied by a given factor k. Given the integer k, return an array answer of size n where answer[i] is the minimum total cost to buy an apple if you start at city i.

import heapq

from collections import defaultdict

import sys

def dijkstra(graph, start, n):

distances = [sys.maxsize] \* (n + 1)

distances[start] = 0

pq = [(0, start)]

while pq:

current\_distance, current\_node = heapq.heappop(pq)

if current\_distance > distances[current\_node]:

continue

for neighbor, weight in graph[current\_node]:

distance = current\_distance + weight

if distance < distances[neighbor]:

distances[neighbor] = distance

heapq.heappush(pq, (distance, neighbor))

return distances

def minimum\_total\_cost(n, roads, appleCost, k):

graph = defaultdict(list)

for a, b, cost in roads:

graph[a].append((b, cost))

graph[b].append((a, cost))

min\_costs = []

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

min\_cost = sys.maxsize

distances = dijkstra(graph, stpart, n)

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

if city != start:

travel\_cost = distances[city]

return\_cost = distances[city] \* k

total\_cost = travel\_cost + return\_cost + appleCost[city - 1]

min\_cost = min(min\_cost, total\_cost)

min\_costs.append(min\_cost)

return min\_costs

n = 4

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

appleCost = [56, 42, 102, 301]

k = 2

print(minimum\_total\_cost(n, roads, appleCost, k))

**Output:**

[54,42,48,51]

1. Customers With Strictly Increasing P

Write an SQL query to report the IDs of the customers with the total purchases strictly increasing yearly. ● The total purchases of a customer in one year is the sum of the prices of their orders in that year. If for some year the customer did not make any order, we consider the total purchases 0. ● The first year to consider for each customer is the year of their first order. ● The last year to consider for each customer is the year of their last order. Return the result table in any order. The query result format

WITH yearly\_purchases AS (

SELECT

customer\_id,

EXTRACT(YEAR FROM order\_date) AS order\_year,

SUM(price) AS yearly\_total

FROM Orders

GROUP BY customer\_id, order\_year

),

ranked\_purchases AS (

SELECT

customer\_id,

order\_year,

yearly\_total,

ROW\_NUMBER() OVER (PARTITION BY customer\_id ORDER BY order\_year) AS row\_num

FROM yearly\_purchases

),

increasing\_check AS (

SELECT

p1.customer\_id,

p1.order\_year,

p1.yearly\_total,

p1.row\_num,

p2.yearly\_total AS prev\_yearly\_total

FROM ranked\_purchases p1

LEFT JOIN ranked\_purchases p2

ON p1.customer\_id = p2.customer\_id AND p1.row\_num = p2.row\_num + 1

)

SELECT DISTINCT customer\_id

FROM increasing\_check

WHERE prev\_yearly\_total IS NULL OR yearly\_total > prev\_yearly\_total

GROUP BY customer\_id

HAVING COUNT(\*) = MAX(row\_num);

**Output:**

**+-------------+**

**| customer\_id |**

**+-------------+**

**| 1 |**

**+-------------+**

1. . Number of Unequal Triplets in Array

You are given a 0-indexed array of positive integers nums. Find the number of triplets (i, j, k) that meet the following conditions: ● 0 <= i < j < k < nums.length ● nums[i], nums[j], and nums[k] are pairwise distinct. ○ In other words, nums[i] != nums[j], nums[i] != nums[k], and nums[j] != nums[k]. Return the number of triplets that meet the condition

from collections import defaultdict

from itertools import combinations

def count\_unequal\_triplets(nums):

freq = defaultdict(int)

for num in nums:

freq[num] += 1

unique\_nums = list(freq.keys())

triplet\_count = 0

for a, b, c in combinations(unique\_nums, 3):

triplet\_count += freq[a] \* freq[b] \* freq[c]

return triplet\_count

nums = [4, 4, 2, 4, 3]

print(count\_unequal\_triplets(nums))

**Output:**

3

1. . Closest Nodes Queries in a Binary Search Tree

You are given the root of a binary search tree and an array queries of size n consisting of positive integers. Find a 2D array answer of size n where answer[i] = [mini, maxi]: ● mini is the largest value in the tree that is smaller than or equal to queries[i]. If a such value does not exist, add -1 instead. ● maxi is the smallest value in the tree that is greater than or equal to queries[i]. If a such value does not exist, add -1 in

import bisect

class TreeNode:

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

self.val = val

self.left = None

self.right = None

def inorder\_traversal(root, result):

if root is not None:

inorder\_traversal(root.left, result)

result.append(root.val)

inorder\_traversal(root.right, result)

def closest\_nodes(root, queries):

sorted\_values = []

inorder\_traversal(root, sorted\_values)

result = []

for q in queries:

pos = bisect.bisect\_left(sorted\_values, q)

if pos < len(sorted\_values) and sorted\_values[pos] == q:

mini = maxi = sorted\_values[pos]

else:

mini = sorted\_values[pos - 1] if pos > 0 else -1

maxi = sorted\_values[pos] if pos < len(sorted\_values) else -1

result.append([mini, maxi])

return result

root = TreeNode(6)

root.left = TreeNode(2)

root.right = TreeNode(13)

root.left.left = TreeNode(1)

root.left.right = TreeNode(4)

root.right.left = TreeNode(9)

root.right.right = TreeNode(15)

root.right.right.left = TreeNode(14)

queries = [2, 5, 16]

print(closest\_nodes(root, queries))

**Output:**

[[2,2],[4,6],[15,-1]]

1. Minimum Fuel Cost to Report to the Capital

There is a tree (i.e., a connected, undirected graph with no cycles) structure country network consisting of n cities numbered from 0 to n - 1 and exactly n - 1 roads. The capital city is city 0. You are given a 2D integer array roads where roads[i] = [ai, bi] denotes that there exists a bidirectional road connecting cities ai and bi. There is a meeting for the representatives of each city. The meeting is in the capital city.There is a car in each city. You are given an integer seats that indicates the number of seats in each car.A representative can use the car in their city to travel or change the car and ride with another representative. The cost of traveling between two cities is one liter of fuel. Return the minimum number of liters of fuel to reach the capital

def minimumFuelCost(roads, seats):

from collections import defaultdict

import math

graph = defaultdict(list)

for a, b in roads:

graph[a].append(b)

graph[b].append(a)

def dfs(node, parent):

representatives = 1

for neighbor in graph[node]:

if neighbor != parent:

child\_reps = dfs(neighbor, node)

trips = math.ceil(child\_reps / seats)

total\_fuel[0] += trips

representatives += child\_reps

return representatives

total\_fuel = [0]

dfs(0, -1)

return total\_fuel[0]

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

seats = 5

print(minimumFuelCost(roads, seats))

**Output:**

3

1. Number of Beautiful Partitions

You are given a string s that consists of the digits '1' to '9' and two integers k and minLength. A partition of s is called beautiful if: ● s is partitioned into k non-intersecting substrings. ● Each substring has a length of at least minLength. ● Each substring starts with a prime digit and ends with a non-prime digit. Prime digits are '2', '3', '5', and '7', and the rest of the digits are non-prime. Return the number of beautiful partitions of s. Since the answer may be very large, return it modulo 109 + 7.A substring is a contiguous sequence of characters within a string

def numberOfBeautifulPartitions(s, k, minLength):

MOD = 10\*\*9 + 7

n = len(s)

prime\_digits = {'2', '3', '5', '7'}

non\_prime\_digits = {'1', '4', '6', '8', '9'}

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

for j in range(minLength - 1, n):

if s[j] in prime\_digits:

dp[1][j] = 1

for i in range(2, k + 1):

for j in range(minLength - 1, n):

if s[j] in non\_prime\_digits:

continue

for l in range(minLength - 1, j + 1):

if s[l] in prime\_digits:

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

result = 0

for j in range(k \* minLength - 1, n):

result = (result + dp[k][j]) % MOD

return result

s = "23542185131"

k = 3

minLength = 2

print(numberOfBeautifulPartitions(s, k, minLength))

**Output:**

3