CSA0674 - DAA ASSIGNMENT 5

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. Note that: • Kelvin = Celsius + 273.15 • Fahrenheit = Celsius * 1.80 + 32.00 Example 1: Input: celsius = 36.50 Output: [309.65000,97.70000] Explanation: Temperature at 36.50 Celsius converted in Kelvin is 309.65 and converted in Fahrenheit is 97.70. Example 2: Input: celsius = 122.11 Output: [395.26000,251.79800] Explanation: Temperature at 122.11 Celsius converted in Kelvin is 395.26 and converted in Fahrenheit is 251.798. Constraints: 0 <= celsius <= 1000

```
def convertTemperature(celsius):
    return [celsius + 273.15, celsius * 1.8 + 32]
celsius = float(input())
print(convertTemperature(celsius))
```

2) 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 elements. Example 1: Input: nums = [3,6,2,7,1], k = 6 Output: 4 Explanation: The subarrays of nums where 6 is the least common multiple of all the subarray's elements are: - [3,6,2,7,1] - [3,6,2,7,1] - [3,6,2,7,1] - [3,6,2,7,1] Example 2:Input: nums = [3], k = 2 Output: 0 Explanation: There are no subarrays of nums where 2 is the least common multiple of all the subarray's elements. Constraints: • 1 <= nums.length <= 1000 • 1

```python
from math import gcd
from functools import reduce

def lcm(a, b):
    return a * b // gcd(a, b)
def numberOfSubarraysWithLCM(nums, k):
    n = len(nums)
    count = 0
    for i in range(n):
        lcm_value = 1
        for j in range(i, n):
            lcm_value = lcm(lcm_value, nums[j])
            if lcm_value == k:
                count += 1
            if lcm_value > k:
                break
    return count
nums = [int(x) for x in input().split()]
k = int(input())
print(numberOfSubarraysWithLCM(nums, k))
```

3) MINIMUM NUMBER OF OPERATINS 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 order. The level of a node is the number of edges along the path between it and the root node. Example 1: Input: root = [1,4,3,7,6,8,5,null,null,null,null,9,null,10] Output: 3 Explanation: - Swap 4 and 3. The 2nd level becomes [3,4]. - Swap 7 and 5. The 3rd level becomes [5,6,8,7]. - Swap 8 and 7. The 3rd level becomes [5,6,7,8]. We used 3 operations so return 3. It can be proven that 3 is the minimum number of operations needed. Example 2: Input: root = [1,3,2,7,6,5,4] Output: 3 Explanation: - Swap 3 and 2. The 2nd level becomes [2,3]. - Swap 7 and 4. The 3rd level becomes [4,6,5,7]. - Swap 6 and 5. The 3rd level becomes [4,5,6,7]. We used 3 operations so return 3. It can be proven that 3 is the minimum number of operations needed. Example 3: Input: root = [1,2,3,4,5,6] Output: 0 Explanation: Each level is already sorted in increasing order so return 0. Constraints: • The number of nodes in the tree is in the range [1, 105]. • 1 <= Node.val <= 105 • All the values of the tree are unique

```python
from collections import deque
def minSwapsToSortLevel(arr):
    n = len(arr)
    sorted_arr = sorted(range(n), key=lambda k: arr[k])
    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 > 1:
            swaps += cycle_size - 1
    return swaps
def minOperationsToSortTree(root):
    if not root:
        return 0
    queue = deque([root])
    total_swaps = 0
    while queue:
        level_size = len(queue)
        level = []
        for _ in range(level_size):
            node = queue.popleft()
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        total_swaps += minSwapsToSortLevel(level)
    return total_swaps
```

4) MAXIMUM NUMBER OF NON OVERLAPPING PALINDROME SUBSTRING :

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. Example 1: Input: s = "abaccdbbd", k = 3 Output: 2 Explanation: We can select the substrings underlined in s = "abaccdbbd". Both "aba" and "dbbd" are palindromes and have a length of at least k = 3. It can be shown that we cannot find a selection with more than two valid substrings. Example 2: Input: s = "adbcda", k = 2 Output: 0 Explanation: There is no palindrome substring of length at least 2 in the string. Constraints: • 1 <= k <= s.length <= 2000 • s consists of lowercase English letters

```
def isPalindrome(s):
    return s == s[::-1]
def maxNonOverlappingPalindromes(s, k):
    n = len(s)
    dp = [0] * (n + 1)
    for i in range(k, n + 1):
        for j in range(i - k, -1, -1):
            if isPalindrome(s[j:i]):
                dp[i] = max(dp[i], dp[j] + 1)
    return dp[n]
s = input().strip()
k = int(input())
print(maxNonOverlappingPalindromes(s, k))
```

5) 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. Example 1: Input: 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 Output: [54,42,48,51] Explanation: The minimum cost for each starting city is the following: - Starting at city 1: You take the path 1 -> 2, buy an apple at city 2, and finally take the path 2 -> 1. The total cost is 4 + 42 + 4 * 2 = 54. - Starting at city 2: You directly buy an apple at city 2. The total cost is 42. - Starting at city 3: You take the path 3 -> 2, buy an apple at city 2, and finally take the path 2 -> 3. The total cost is 2 + 42 + 2 * 2 = 48. - Starting at city 4: You take the path 4 -> 3 -> 2 then you buy at city 2, and finally take the path 2 -> 3 -> 4. The total cost is 1 + 2 + 42 + 1 * 2 + 2 * 2 = 51. Example 2: Input: n = 3, roads = [[1,2,5],[2,3,1],[3,1,2]], appleCost = [2,3,1], k = 3 Output: [2,3,1] Explanation: It is always optimal to buy the

apple in the starting city. Constraints: • 2 <= n <= 1000 • 1 <= roads.length <= 1000 • 1 <= ai, bi <= n • ai != bi • 1 <= costi <= 105 • appleCost.length == n • 1 <= appleCost[i] <= 105 • 1 <= k <= 100 • There are no repeated edges

```
import heapq
import math
def minCostToBuyApples(n, roads, appleCost, k):
    graph = [[] for _ in range(n)]
    for a, b, cost in roads:
        graph[a-1].append((b-1, cost))
        graph[b-1].append((a-1, cost))
    def dijkstra(start):
        dist = [math.inf] * n
        dist[start] = 0
        pq = [(0, start)]
        while pq:
            d, u = heapq.heappop(pq)
            if d > dist[u]:
                continue
            for v, cost in graph[u]:
                if dist[u] + cost < dist[v]:
                    dist[v] = dist[u] + cost
                    heapq.heappush(pq, (dist[v], v))
        return dist
    answer = []
    for i in range(n):
        dist = dijkstra(i)
        min_cost = min(appleCost[j] + dist[j] + dist[j] * k for j in range(n))
        answer.append(min_cost)
    return answer
n = int(input())
roads = [list(map(int, input().split())) for _ in range(n-1)]
appleCost = list(map(int, input().split()))
k = int(input())
print(minCostToBuyApples(n, roads, appleCost, k))
```

6) CUSTOMERS WITH STRICTLY INCREASING PURCHASES :

Table: Orders +--------------+------+ | Column Name | Type | +--------------+------+ | order_id | int | | customer_id | int | | order_date | date | | price | int | +--------------+------+ order_id is the primary key for this table. Each row contains the id of an order, the id of customer that ordered it, the date of the order, and its price. 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 is in the following example. Example 1: Input: Orders table: +----------+------------+------------+-------+ | order_id | customer_id | order_date | price | +----------+------------+------------+-------+ | 1 | 1 | 2019-07-01 | 1100 | | 2 | 1 | 2019-11-01 | 1200 | | 3 | 1 | 2020-05-26 | 3000 | | 4 | 1 | 2021-08-31 | 3100

| | 5 | 1 | 2022-12-07 | 4700 | | 6 | 2 | 2015-01-01 | 700 | | 7 | 2 | 2017-11-07 | 1000 | | 8 | 3 | 2017-01-01 | 900 | | 9 | 3 | 2018-11-07 | 900 | +-------+-------------+-----------+--------+ Output: +-------------+ | customer_id | +-------------+ | 1 | +-------------+ Explanation: Customer 1: The first year is 2019 and the last year is 2022 - 2019: 1100 + 1200 = 2300 - 2020: 3000 - 2021: 3100 - 2022: 4700 We can see that the total purchases are strictly increasing yearly, so we include customer 1 in the answer. Customer 2: The first year is 2015 and the last year is 2017 - 2015: 700 - 2016: 0 - 2017: 1000 We do not include customer 2 in the answer because the total purchases are not strictly increasing. Note that customer 2 did not make any purchases in 2016. Customer 3: The first year is 2017, and the last year is 2018 - 2017: 900 - 2018: 900 We can see that the total purchases are strictly increasing yearly, so we include customer 1 in the answe

```
SELECT customer_id
FROM Orders
GROUP BY customer_id
HAVING MIN(order_date) = YEAR(order_date)
   AND (SUM(price) > 0 AND SUM(price) = ALL(
    SELECT SUM(price)
    FROM Orders o
    WHERE o.customer_id = Orders.customer_id
    GROUP BY YEAR(order_date)
    ORDER BY YEAR(order_date) ASC
));
```

7) NUMBER OF UNEQUAL TRIPLETS IN AN 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 conditions. Example 1: Input: nums = [4,4,2,4,3] Output: 3 Explanation: The following triplets meet the conditions: - (0, 2, 4) because 4 != 2 != 3 - (1, 2, 4) because 4 != 2 != 3 - (2, 3, 4) because 2 != 4 != 3 Since there are 3 triplets, we return 3. Note that (2, 0, 4) is not a valid triplet because

2 > 0. Example 2: Input: nums = [1,1,1,1,1] Output: 0 Explanation: No triplets meet the conditions so we return 0. Constraints: • 3 <= nums.length <= 100 • 1 <= nums[i] <= 100

```
def unequalTriplets(nums):
    n = len(nums)
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if nums[i] != nums[j] and nums[i] != nums[k] and nums[j] != nums[k]:
                    count += 1
    return count
nums = list(map(int, input().split()))
print(unequalTriplets(nums))
```

8) 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 instead. Return the array answer. Example 1: Input: root = [6,2,13,1,4,9,15,null,null,null,null,null,null,14],

queries = [2,5,16] Output: [[2,2],[4,6],[15,-1]] Explanation: We answer the queries in the following way: - The largest number that is smaller or equal than 2 in the tree is 2, and the smallest number that is greater or equal than 2 is still 2. So the answer for the first query is [2,2]. - The largest number that is smaller or equal than 5 in the tree is 4, and the smallest number that is greater or equal than 5 is 6. So the answer for the second query is [4,6]. - The largest number that is smaller or equal than 16 in the tree is 15, and the smallest number that is greater or equal than 16 does not exist. So the answer for the third query is [15,-1]. Example 2: Input: root = [4,null,9], queries = [3] Output: [[-1,4]] Explanation: The largest number that is smaller or equal to 3 in the tree does not exist, and the smallest number that is greater or equal to 3 is 4. So the answer for the query is [-1,4]. Constraints: • The number of nodes in the tree is in the range [2, 105]. • 1 <= Node.val <= 106 • n == queries.length • 1 <= n <= 105 • 1 <= queries

```python
def closestNodesQueries(root, queries):
    def inorder(node):
        if not node:
            return []
        return inorder(node.left) + [node.val] + inorder(node.right)

    sorted_nodes = inorder(root)
    answer = []
    for q in queries:
        mini = -1
        maxi = -1
        for val in sorted_nodes:
            if val <= q:
                mini = val
            if val >= q:
                maxi = val
                break
        answer.append([mini, maxi])
    return answer
```

9) 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 city. Example 1: Input: roads = [[0,1],[0,2],[0,3]], seats = 5 Output: 3 Explanation: - Representative1 goes directly to the capital with 1 liter of fuel. - Representative2 goes directly to the capital with 1 liter of fuel. - Representative3 goes directly to the capital with 1 liter of fuel. It costs 3 liters of fuel at minimum. It can be proven that 3 is the minimum number of liters of fuel needed. Example 2: Input: roads = [[3,1],[3,2],[1,0],[0,4],[0,5],[4,6]], seats = 2 Output: 7 Explanation: - Representative2 goes directly to city 3 with 1 liter of fuel. - Representative2 and representative3 go together to city 1 with 1 liter of fuel. - Representative2 and representative3 go together to the capital with 1 liter of fuel. - Representative1 goes directly to the capital with 1 liter of fuel. - Representative5 goes directly to the capital with 1 liter of fuel. -

Representative6 goes directly to city 4 with 1 liter of fuel. - Representative4 and representative6 go together to the capital with 1 liter of fuel. It costs 7 liters of fuel at minimum. It can be proven that 7 is the minimum number of liters of fuel needed. Example 3: Input: roads = [], seats = 1 Output: 0 Explanation: No representatives need to travel to the capital city. Constraints: • 1 <= n <= 105 • roads.length == n - 1 • roads[i].length == 2 • 0 <= ai, bi < n • ai != bi • roads represents a valid tree. • 1 <=

```python
from collections import defaultdict, deque
def minimumFuelCost(n, roads, seats):
    graph = defaultdict(list)
    for a, b in roads:
        graph[a].append(b)
        graph[b].append(a)
    visited = [False] * n
    representatives = [0] * n
    fuel = 0
    def dfs(city):
        nonlocal fuel
        visited[city] = True
        rep = 1
        for neighbor in graph[city]:
            if not visited[neighbor]:
                rep += dfs(neighbor)
        if city != 0:
            fuel += (rep + seats - 1) // seats
        return rep
    dfs(0)
    return fuel
n = int(input())
roads = [list(map(int, input().split())) for _ in range(n-1)]
seats = int(input())
print(minimumFuelCost(n, roads, seats))
```

10) 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. Example 1: Input: s = "23542185131", k = 3, minLength = 2 Output: 3 Explanation: There exists three ways to create a beautiful partition: "2354 | 218 | 5131" "2354 | 21851 | 31" "2354218 | 51 | 31" Example 2: Input: s = "23542185131", k = 3, minLength = 3 Output: 1 Explanation: There exists one way to create a beautiful partition: "2354 | 218 | 5131". Example 3: Input: s = "3312958", k = 3, minLength = 1 Output: 1 Explanation: There exists one way to create a beautiful partition: "331 | 29 | 58". Constraints: • 1 <= k, minLength <= s.length <= 1000 • s consists of the digits '1' to '9'

```python
def isPrime(digit):
    return digit in '2357'
def beautifulPartitions(s, k, minLength):
    MOD = 10**9 + 7
    n = len(s)
    if k * minLength > n:
        return 0
    dp = [[0] * (k + 1) for _ in range(n + 1)]
    dp[0][0] = 1
    for i in range(minLength, n + 1):
        for j in range(1, k + 1):
            for l in range(i - minLength, -1, -1):
                if isPrime(s[l]) and not isPrime(s[i-1]):
                    dp[i][j] = (dp[i][j] + dp[l][j-1]) % MOD
    return dp[n][k]
s = input().strip()
k = int(input())
minLength = int(input())
print(beautifulPartitions(s, k, minLength))
```