ASSIGNMENT-7

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

**PROGRAM**

def convert\_temperature(celsius):

kelvin = celsius + 273.15

fahrenheit = celsius \* 1.80 + 32.00

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

celsius = 36.50

print(convert\_temperature(celsius))

2. 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.

**PROGRAM**

import math

from functools import reduce

def lcm(x, y):

return abs(x \* y) // math.gcd(x, y)

def lcm\_of\_list(lst):

return reduce(lcm, lst, 1)

def subarrays\_with\_lcm(nums, k):

count = 0

for i in range(len(nums)):

for j in range(i, len(nums)):

subarray\_lcm = lcm\_of\_list(nums[i:j+1])

if subarray\_lcm == k:

count += 1

return count

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

k = 6

print(subarrays\_with\_lcm(nums, k))

3. 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 order.

**PROGRAM**

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def min\_operations\_to\_sort\_levels(root):

from collections import deque

if not root:

return 0

def level\_order\_traversal(root):

levels = []

queue = deque([root])

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)

levels.append(level)

return levels

def min\_swaps\_to\_sort(arr):

n = len(arr)

sorted\_arr = sorted(arr)

index\_map = {value: idx for idx, value in enumerate(arr)}

swaps = 0

for i in range(n):

if arr[i] != sorted\_arr[i]:

swaps += 1

swap\_idx = index\_map[sorted\_arr[i]]

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

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

return swaps

levels = level\_order\_traversal(root)

total\_swaps = 0

for level in levels:

total\_swaps += min\_swaps\_to\_sort(level)

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.right.left.left = TreeNode(9)

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

print(min\_operations\_to\_sort\_levels(root))

4. 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.

**PROGRAM**

def max\_palindrome\_substrings(s, k):

def is\_palindrome(sub):

return sub == sub[::-1]

n = len(s)

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

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

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

j = i + length

if is\_palindrome(s[i:j]):

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

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

return max(max(row) for row in dp)

s = "abaccdbbd"

k = 3

print(max\_palindrome\_substrings(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

**PROGRAM**

import heapq

def min\_cost\_to\_buy\_apples(n, roads, appleCost, k):

graph = {i: [] for i in range(1, n + 1)}

for u, v, cost in roads:

graph[u].append((v, cost))

graph[v].append((u, cost))

def dijkstra(start):

heap = [(0, start)]

dist = {i: float('inf') for i in range(1, n + 1)}

dist[start] = 0

while heap:

current\_dist, node = heapq.heappop(heap)

if current\_dist > dist[node]:

continue

for neighbor, weight in graph[node]:

distance = current\_dist + weight

if distance < dist[neighbor]:

dist[neighbor] = distance

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

return dist

min\_cost = []

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

dist = dijkstra(i)

min\_cost\_i = float('inf')

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

if i != j:

min\_cost\_i = min(min\_cost\_i, dist[j] + appleCost[j - 1] + dist[j] \* k)

else:

min\_cost\_i = min(min\_cost\_i, appleCost[j - 1])

min\_cost.append(min\_cost\_i)

return min\_cost

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(min\_cost\_to\_buy\_apples(n, roads, appleCost, k))

7. 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 conditions.

**PROGRAM**

def count\_unequal\_triplets(nums):

count = 0

n = len(nums)

for i in range(n - 2):

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

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

print(count\_unequal\_triplets(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]:

**PROGRAM**

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def inorder\_traversal(root):

result = []

stack = []

current = root

while stack or current:

while current:

stack.append(current)

current = current.left

current = stack.pop()

result.append(current.val)

current = current.right

 return result

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.

**PROGRAM**

from collections import defaultdict

import math

def minimumFuelCost(roads, seats):

n = len(roads) + 1

tree = defaultdict(list)

for a, b in roads:

tree[a].append(b)

tree[b].append(a)

def dfs(node, parent):

total\_representatives = 1

fuel = 0

for neighbor in tree[node]:

if neighbor != parent:

sub\_representatives, sub\_fuel = dfs(neighbor, node)

total\_representatives += sub\_representatives

fuel += sub\_fuel

if node != 0:

cars\_needed = math.ceil(total\_representatives / seats)

fuel += cars\_needed

return total\_representatives, fuel

\_, total\_fuel = dfs(0, -1)

return total\_fuel

print(minimumFuelCost([[0, 1], [0, 2], [0, 3]], 5))

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.

**PROGRAM**

def is\_prime\_digit(ch):

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

def num\_beautiful\_partitions(s, k, minLength):

n = len(s)

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

dp[0][0] = 1

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

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

for l in range(minLength, i + 1):

if is\_prime\_digit(s[i - l]) and not is\_prime\_digit(s[i - 1]):

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

return dp[n][k]

print(num\_beautiful\_partitions("23542185131", 3, 2))