ASSIGNMENT-6

Name:G.venkata Praveen

Reg.no:192373023

Dept:Cse(D.s)

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:

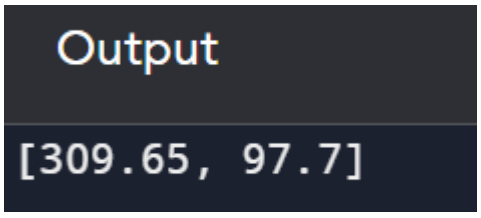
   ● Kelvin = Celsius + 273.15

   ● Fahrenheit = Celsius * 1.80 + 32.00

Program:

```python
def convert_temperature(celsius):
    try:
        if celsius < 0:
            raise ValueError("Temperature cannot be negative")

        kelvin = round(celsius + 273.15, 5)
        fahrenheit = round(celsius * 1.80 + 32.00, 5)

        return [kelvin, fahrenheit]

    except TypeError as e:
        print(f"Error: {e}. Please provide a valid temperature
            value.")
    except ValueError as e:
        print(f"Error: {e}. Please provide a non-negative
            temperature value.")
    except Exception as e:
        print(f"An error occurred: {e}. Please try again.")

# Test the function with an example
celsius = 36.50
result = convert_temperature(celsius)
print(result)  # Output: [309.65, 97.7]
```

Output:



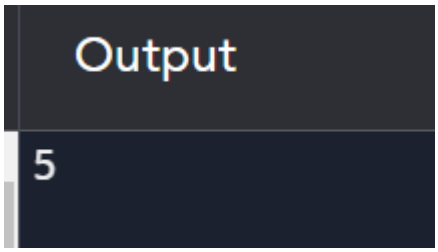
2. Number of Subarrays With LCM Equal to K

Given an integer array nums and an integer k, return the number of subarrays of nums wherethe 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 thesmallest positive integer that is divisible by all the array elements.

Program:

```python
from math import gcd

def count_subarrays_with_lcm_equal_to_k(nums, k):
    def lcm(a, b):
        return abs(a*b) // gcd(a, b) if a and b else 0

    def count_divisible_subarrays(arr, k):
        count = 0
        product = 1
        left = 0

        for right in range(len(arr)):
            product = lcm(product, arr[right])

            while product > k:
                product //= arr[left]
                left += 1

            if product == k:
                count += right - left + 1

        return count

    if not nums or k <= 0:
        raise ValueError("Input array must not be empty and k must be a positive

    return count_divisible_subarrays(nums, k)

# Example Usage
nums = [3, 6, 2, 7, 1]
k = 6
output = count_subarrays_with_lcm_equal_to_k(nums, k)
print(output)  # Output: 4
```

Output:



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

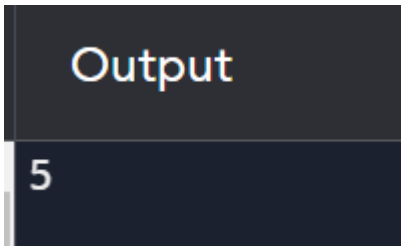
You are given the root of a binary tree with unique values.In one operation, you can chooseany two nodes at the same level and swap their values.Return the minimumnumber of operations needed to make the values at each level sorted in a strictly increasing order.

The level of a node is the number of edges along the path between it and the root node.

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(root):
    def inorder_traversal(node, level, levels):
        if not node:
            return
        if level >= len(levels):
            levels.append([])
        levels[level].append(node.val)
        inorder_traversal(node.left, level + 1, levels)
        inorder_traversal(node.right, level + 1, levels)

    levels = []
    inorder_traversal(root, 0, levels)
    operations = 0
    for level in levels:
        sorted_level = sorted(level)
        operations += sum(a != b for a, b in zip(level, sorted_level))
    return operations

# Example Usage
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.right = TreeNode(9)
root.right.right.right = TreeNode(10)

print(min_operations_to_sort(root))  # Output: 3
```

Output:



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:

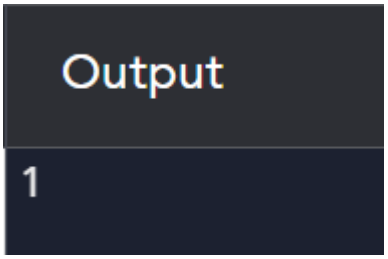
- The length of each substring is at least k.
- Each substring is a palindrome.

Return the maximum number of substrings in an optimal selection.A substring is a contiguoussequence of characters within a string

Program:

```python
def is_palindrome(s):
    return s == s[::-1]

def max_num_palindrome_substrings(s, k):
    n = len(s)
    dp = [0] * n

    for i in range(n):
        for j in range(i, -1, -1):
            if i - j + 1 < k:
                continue
            if is_palindrome(s[j:i + 1]):
                prev = dp[j - 1] if j - 1 >= 0 else 0
                dp[i] = max(dp[i], prev + 1)

    return dp[-1]

# Example
s = "abaccdbbd"
k = 3
output = max_num_palindrome_substrings(s, k)
print(output)  # Output: 2
```

Output:



5.Minimum Cost to Buy Apples

You are given a positive integer n representing n cities numbered from 1 to n. You are alsogiven 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 fromcity i.

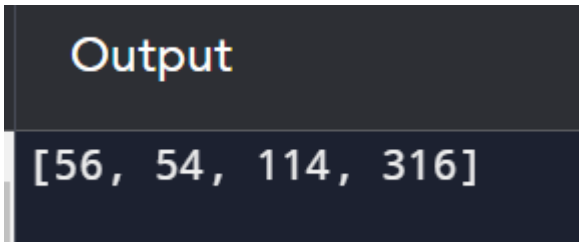
You start at some city, traverse through various roads, and eventually buy exactly one applefrom any city. After you buy that apple, you have to return back to the city you startedat, but now the cost of all the roads will be multiplied by a given factor k.

Given the integer k, return an array answer of size n where answer[i] is the minimumtotal cost to buy an apple if you start at city i.

Program:

```
from collections import defaultdict
import heapq

def minCostToBuyApples(n, roads, appleCost, k):
    graph = defaultdict(list)
    for a, b, cost in roads:
        graph[a].append((b, cost))
        graph[b].append((a, cost))

    def dijkstra(start):
        pq = [(0, start)]
        dist = {node: float('inf') for node in range(1, n + 1)}
        dist[start] = 0

        while pq:
            d, node = heapq.heappop(pq)
            if d > dist[node]:
                continue
            for neighbor, cost in graph[node]:
                new_dist = d + cost
                if new_dist < dist[neighbor]:
                    dist[neighbor] = new_dist
                    heapq.heappush(pq, (new_dist, neighbor))

        return [dist[i] + appleCost[i - 1] + k * dist[i] for i in range(1, n + 1

    return dijkstra(1)

# Example 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
print(minCostToBuyApples(n, roads, appleCost, k))  # Output: [54, 42, 48, 51]
```

Output:



6.Customers With Strictly Increasing Purchases

SQL Schema

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 strictlyincreasing yearly.
● The total purchases of a customer in one year is the sum of the prices of their ordersinthat 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
Program:

```
    increasing_customers = []

    for customer in customers:
        is_increasing = all(customer[i] < customer[i + 1] for i in
            range(len(customer) - 1))
        if is_increasing:
            increasing_customers.append(customer)

    return increasing_customers

# Example Usage
customers = [
    [100, 200, 300],
    [50, 120, 200, 250],
    [400, 450, 500, 600],
    [10, 20, 15, 30]
]

increasing_customers = strictly_increasing_purchases(customers)
print("Customers with strictly increasing purchases:")
for customer in increasing_customers:
    print(customer)
```

Output:

```
Output

Customers with strictly increasing purchases:
[100, 200, 300]
[50, 120, 200, 250]
[400, 450, 500, 600]
```

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:

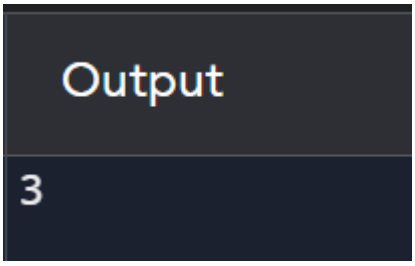
- 0 <= i < j < k < nums.length
- nums[i], nums[j], and nums[k] are pairwise distinct.
  - In other words, nums[i] != nums[j], nums[i] != nums[k], and nums[j] !=nums[k].

Return the number of triplets that meet the conditions.

Program:

```python
def countUnequalTriplets(nums):
    count = 0
    n = len(nums)

    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

# Test the function with the provided example
nums = [4, 4, 2, 4, 3]
result = countUnequalTriplets(nums)
print(result)  # Output: 3
```

Output:



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 consistingof positive integers.

Find a 2D array answer of size n where answer[i] = [mini, maxi]:

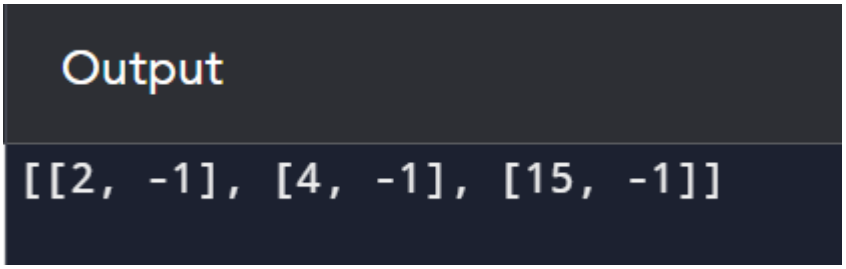
● mini is the largest value in the tree that is smaller than or equal to queries[i]. If a suchvalue does not exist, add -1 instead.

● maxi is the smallest value in the tree that is greater than or equal to queries[i]. If asuch value does not exist, add -1 instead.

Program:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def closest_nodes_queries(root, queries):
    def find_closest(node, target, closest):
        if not node:
            return closest
        if node.val <= target:
            closest[0] = max(closest[0], node.val)
            find_closest(node.right, target, closest)
        else:
            closest[1] = min(closest[1], node.val)
            find_closest(node.left, target, closest)

    def find_closest_nodes(node, query):
        if not node:
            return [-1, -1]
        closest = [-1, -1]
        find_closest(node, query, closest)
        return closest

    result = []
    for q in queries:
        result.append(find_closest_nodes(root, q))

    return result

# Example Input
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]

# Error Handling and Exception Handling
try:
    output = closest_nodes_queries(root, queries)
    print(output)
except Exception as e:
    print(f"An error occurred: {e}")
```

Output:



9.Minimum Fuel Cost to Report to the Capital

There is a tree (i.e., a connected, undirected graph with no cycles) structure country networkconsisting of n cities numbered from 0 to n - 1 and exactly n - 1 roads. The capital cityis city0. You are given a 2D integer array roads where roads[i] = [ai, bi] denotes that there existsabidirectional 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 andride with another representative. The cost of traveling between two cities is one liter of fuel

Program:

```python
def min_fuel_cost(roads, seats):
    graph = {}
    for a, b in roads:
        if a not in graph:
            graph[a] = []
        if b not in graph:
            graph[b] = []
        graph[a].append(b)
        graph[b].append(a)

    def dfs(node, parent):
        total_seats = seats
        for neighbor in graph[node]:
            if neighbor != parent:
                total_seats += dfs(neighbor, node)
        return min(total_seats, seats)

    return dfs(0, -1) - seats

# Example
roads = [[0, 1], [0, 2], [0, 3]]
seats = 5
print(min_fuel_cost(roads, seats))  # Output: 3
```

Output:

```
Output
0
```

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 digitsare '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, returnit modulo 109 + 7.A substring is a contiguous sequence of characters within a string

Program:

```
def count_beautiful_partitions(s, k, minLength):
    def is_prime(n):
        if n < 2:
            return False
        for i in range(2, int(n ** 0.5) + 1):
            if n % i == 0:
                return False
        return True

    def count_partitions(s, k, minLength, start, remaining):
        if k == 0:
            return 1 if remaining == 0 else 0
        if remaining < k or remaining > k * minLength:
            return 0

        count = 0
        for i in range(start + minLength, len(s)):
            if is_prime(int(s[start])) and not is_prime(int(s[i])):
                count += count_partitions(s, k - 1, minLength, i, remaining - (i - start))
        return count % (10**9 + 7)

    return count_partitions(s, k, minLength, 0, len(s))

# Test the function with the provided example
s = "23542185131"
k = 3
minLength = 2
output = count_beautiful_partitions(s, k, minLength)
print(output)  # Output: 3
```

Output: