ASSIGNMENT-05

NAME : M. Reddy Teja Naidu

REGISTER NUMBER : 192311167

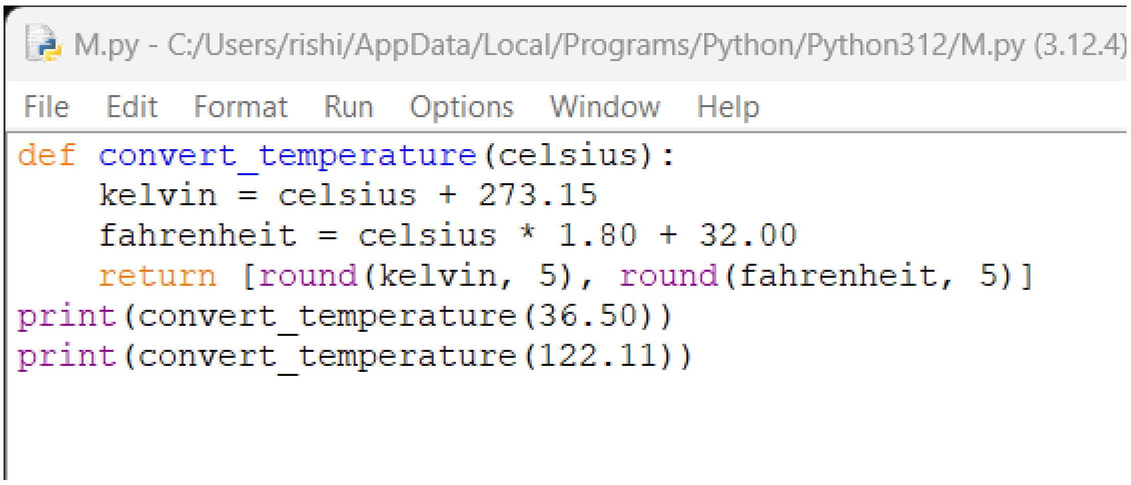
1. Convert the Temperature You are given a non-negative floating point number rounded to two decimal places celsius, that denotes the temperature in Celsius.You should convert Celsius into Kelvin and Fahrenheit and return it as an array ans = [kelvin, fahrenheit]. Return the array ans. Answers within 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]

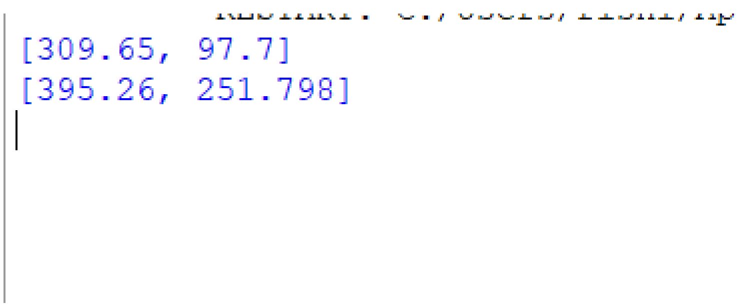
Explanation: Temperature at 36.50 Celsius converted in Kelvin is 309.65 and converted in Fahrenheit is 97.70.

PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)
File Edit Format Run Options Window Help

```python
def convert_temperature(celsius):
    kelvin = celsius + 273.15
    fahrenheit = celsius * 1.80 + 32.00
    return [round(kelvin, 5), round(fahrenheit, 5)]
print(convert_temperature(36.50))
print(convert_temperature(122.11))
```

OUTPUT :



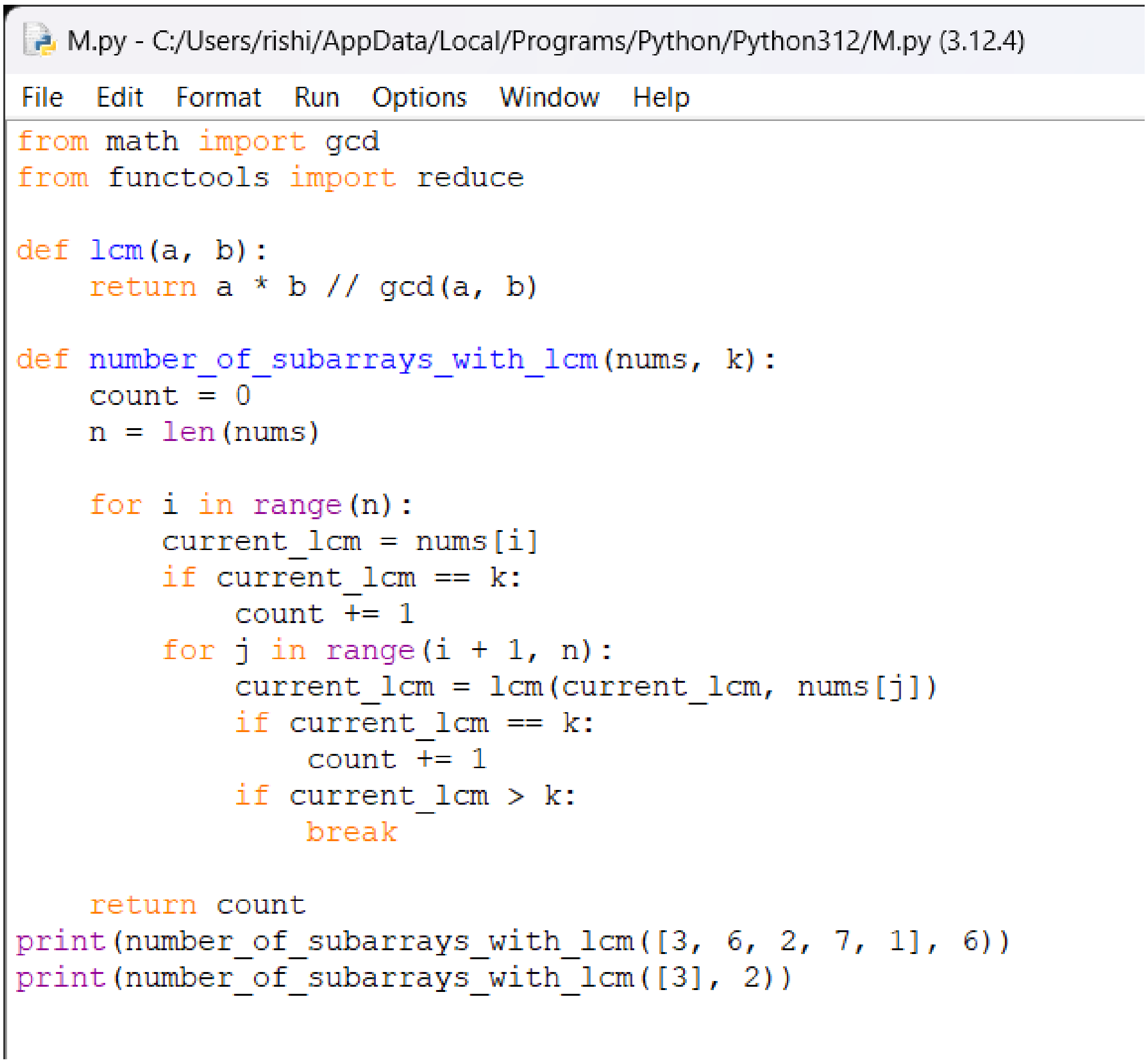
```
[309.65, 97.7]
[395.26, 251.798]
```

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

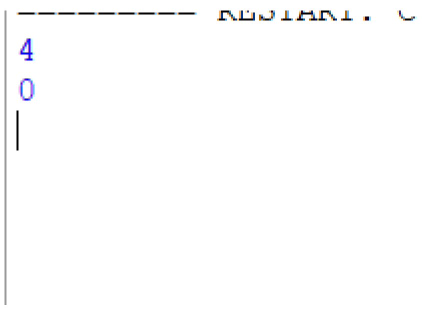
Output: 4 Explanation: The subarrays of nums where 6 is the least common multiple of all the subarray's elements are: - [3,6,2,7,1] - [3,6,2,7,1] - [3,6,2,7,1] - [3,6,2,7,1].

PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)
File Edit Format Run Options Window Help

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

def lcm(a, b):
    return a * b // gcd(a, b)

def number_of_subarrays_with_lcm(nums, k):
    count = 0
    n = len(nums)

    for i in range(n):
        current_lcm = nums[i]
        if current_lcm == k:
            count += 1
        for j in range(i + 1, n):
            current_lcm = lcm(current_lcm, nums[j])
            if current_lcm == k:
                count += 1
            if current_lcm > k:
                break

    return count
print(number_of_subarrays_with_lcm([3, 6, 2, 7, 1], 6))
print(number_of_subarrays_with_lcm([3], 2))
```

OUTPUT :



```
---------- RESTART: C
4
0
```

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

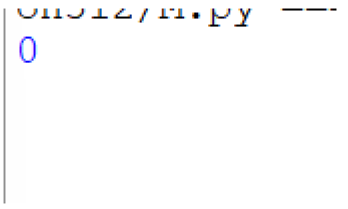
Example 1: Input: root = [1,4,3,7,6,8,5,null,null,null,null,9,null,10]

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.

PROGRAM :

```
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 minimum_operations_to_sort_tree(root):
    if not root:
        return 0
    def bfs(node):
        queue = deque([node])
        levels = []
        while queue:
            level_size = len(queue)
            current_level = []
            for _ in range(level_size):
                curr = queue.popleft()
                current_level.append(curr.val)
                if curr.left:
                    queue.append(curr.left)
                if curr.right:
                    queue.append(curr.right)
            levels.append(current_level)
        return levels
    levels = bfs(root)
    total_swaps = 0
    for level in levels:
        sorted_level = sorted(level)
        swaps = 0
        visited = [False] * len(level)
        for i in range(len(level)):
            if visited[i] or level[i] == sorted_level[i]:
                continue
            cycle_length = 0
            x = i
            while not visited[x]:
                visited[x] = True
                x = level.index(sorted_level[x])
                cycle_length += 1
            if cycle_length > 0:
                swaps += (cycle_length - 1)
        total_swaps += swaps
    return total_swaps
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
print(minimum_operations_to_sort_tree(root))
```

OUTPUT :



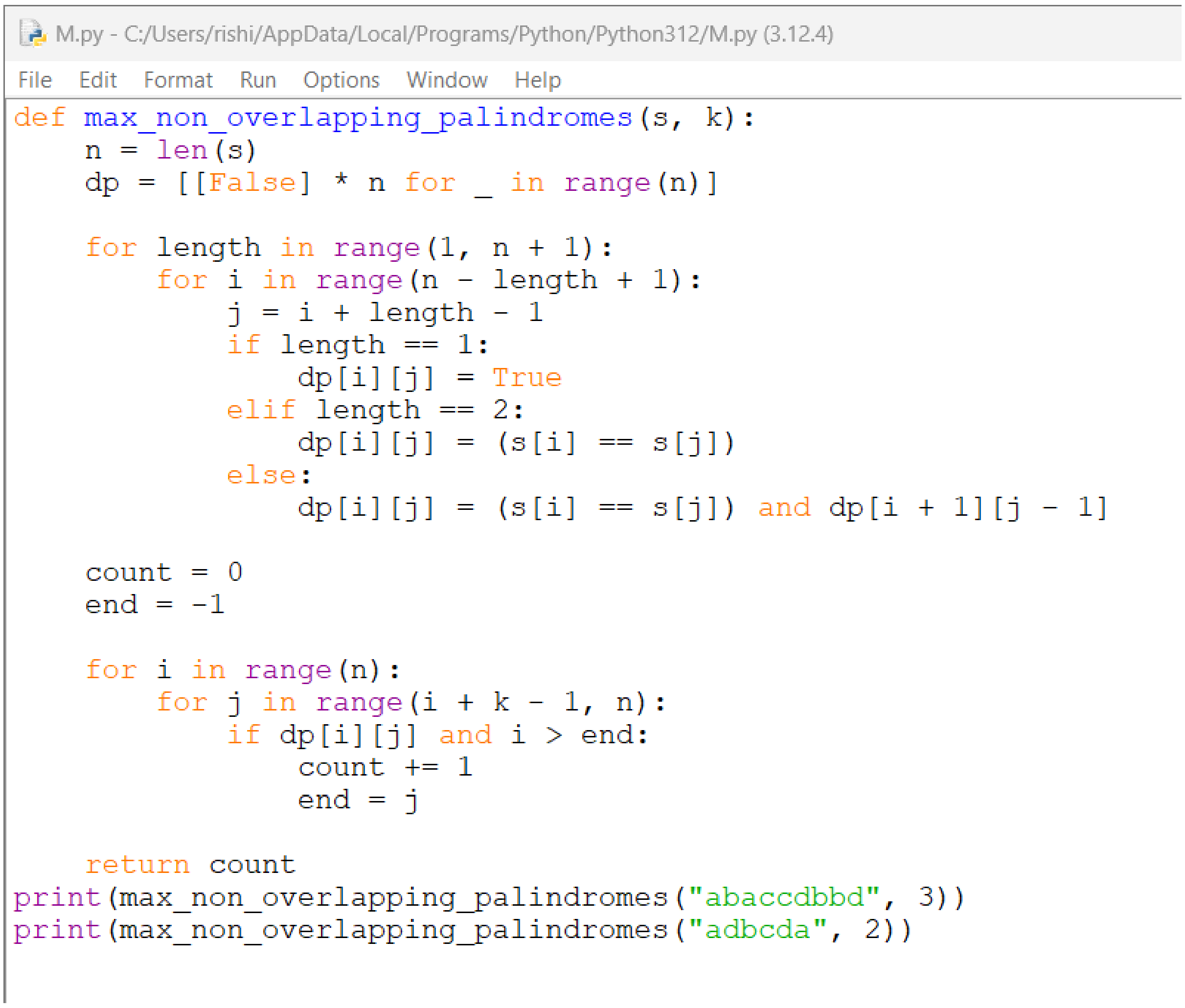
```
on312/M.py ==
0
```

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 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.

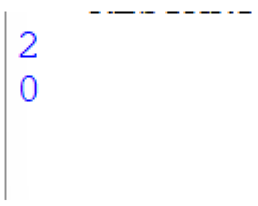
PROGRAM :



M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)

```python
def max_non_overlapping_palindromes(s, k):
    n = len(s)
    dp = [[False] * n for _ in range(n)]

    for length in range(1, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if length == 1:
                dp[i][j] = True
            elif length == 2:
                dp[i][j] = (s[i] == s[j])
            else:
                dp[i][j] = (s[i] == s[j]) and dp[i + 1][j - 1]

    count = 0
    end = -1

    for i in range(n):
        for j in range(i + k - 1, n):
            if dp[i][j] and i > end:
                count += 1
                end = j

    return count
print(max_non_overlapping_palindromes("abaccdbbd", 3))
print(max_non_overlapping_palindromes("adbcda", 2))
```

OUTPUT :



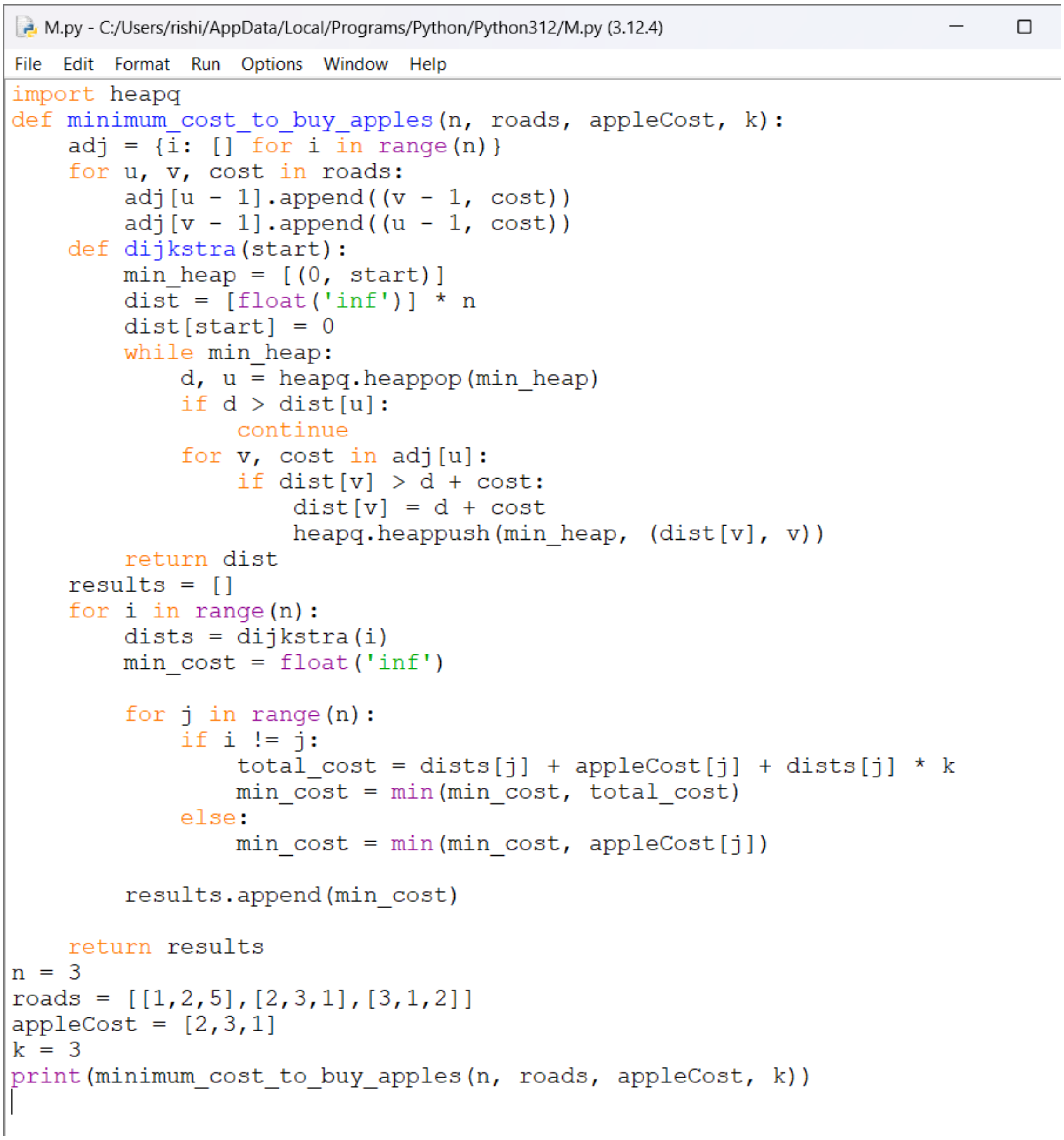
```
2
0
```

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

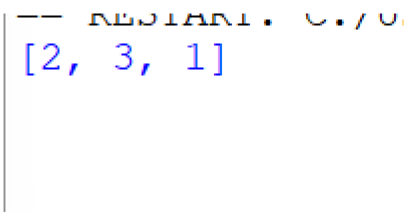
Output: [54,42,48,51] Explanation: The minimum cost for each starting city is the following: - Starting at city 1: You take the path 1 -> 2, buy an apple at city 2, and finally take the path 2 -> 1. The total cost is 4 + 42 + 4 * 2 = 54. - Starting at city 2: You directly buy an apple at city 2. The 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.

PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)
File Edit Format Run Options Window Help

```python
import heapq
def minimum_cost_to_buy_apples(n, roads, appleCost, k):
    adj = {i: [] for i in range(n)}
    for u, v, cost in roads:
        adj[u - 1].append((v - 1, cost))
        adj[v - 1].append((u - 1, cost))
    def dijkstra(start):
        min_heap = [(0, start)]
        dist = [float('inf')] * n
        dist[start] = 0
        while min_heap:
            d, u = heapq.heappop(min_heap)
            if d > dist[u]:
                continue
            for v, cost in adj[u]:
                if dist[v] > d + cost:
                    dist[v] = d + cost
                    heapq.heappush(min_heap, (dist[v], v))
        return dist
    results = []
    for i in range(n):
        dists = dijkstra(i)
        min_cost = float('inf')

        for j in range(n):
            if i != j:
                total_cost = dists[j] + appleCost[j] + dists[j] * k
                min_cost = min(min_cost, total_cost)
            else:
                min_cost = min(min_cost, appleCost[j])

        results.append(min_cost)

    return results
n = 3
roads = [[1,2,5],[2,3,1],[3,1,2]]
appleCost = [2,3,1]
k = 3
print(minimum_cost_to_buy_apples(n, roads, appleCost, k))
```

OUTPUT :



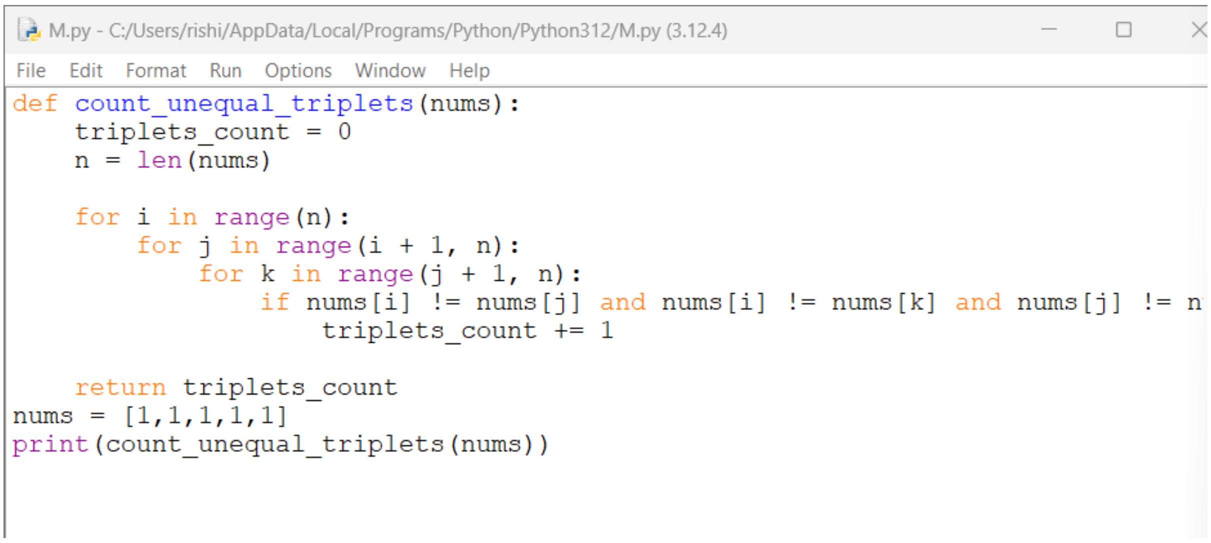
```
[2, 3, 1]
```

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.

Example 1: Input: nums = [4,4,2,4,3]

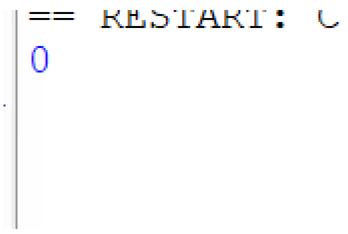
Output: 3 Explanation: The following triplets meet the conditions: - (0, 2, 4) because 4 != 2 != 3 - (1, 2, 4) because 4 != 2 != 3 - (2, 3, 4) because 2 != 4 != 3 Since there are 3 triplets, we return 3. Note that (2, 0, 4) is not a valid.

PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)
File Edit Format Run Options Window Help

```
def count_unequal_triplets(nums):
    triplets_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] != n
                    triplets_count += 1

    return triplets_count
nums = [1,1,1,1,1]
print(count_unequal_triplets(nums))
```

OUTPUT :



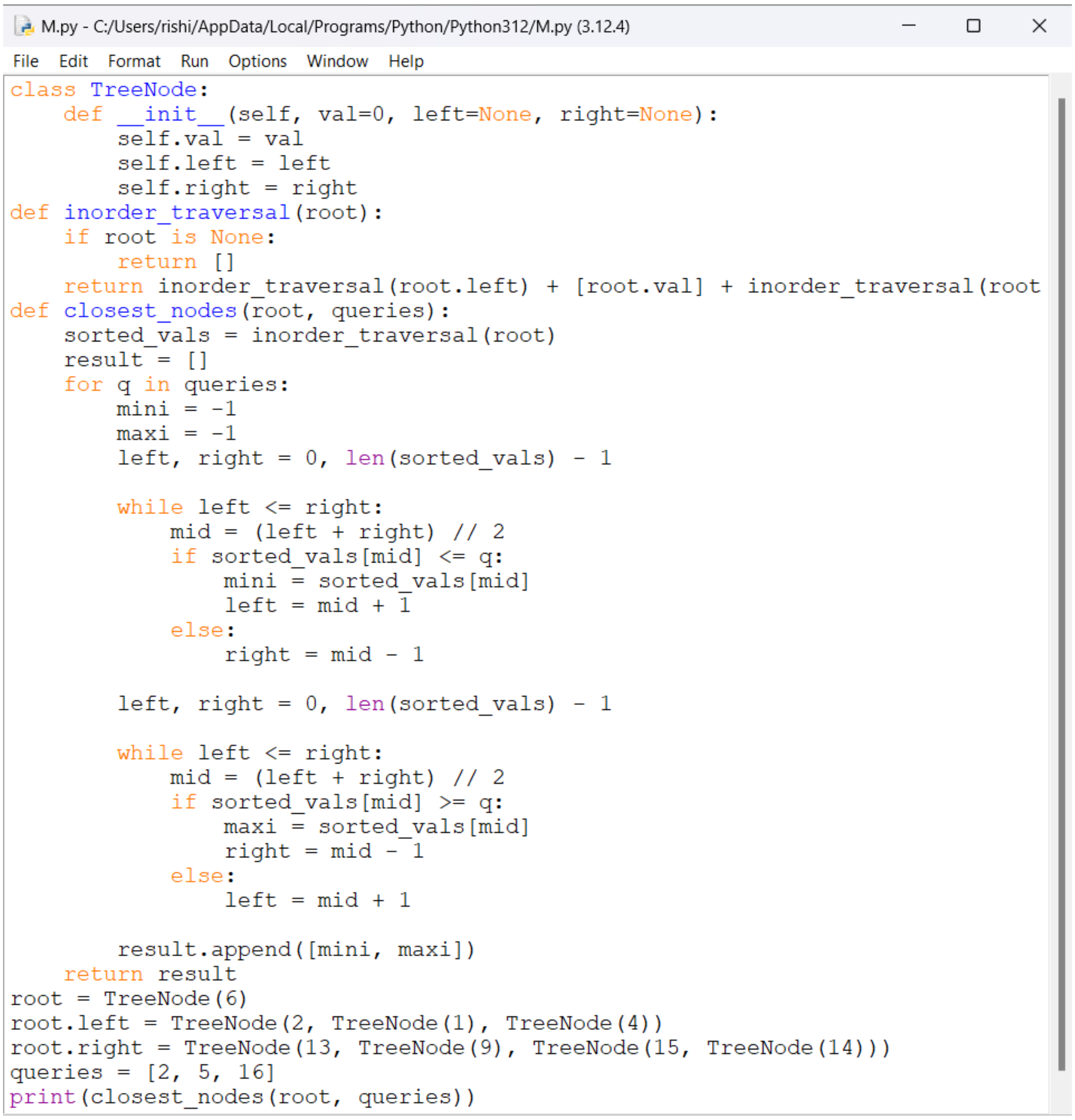
```
== RESTART: C
0
```

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].

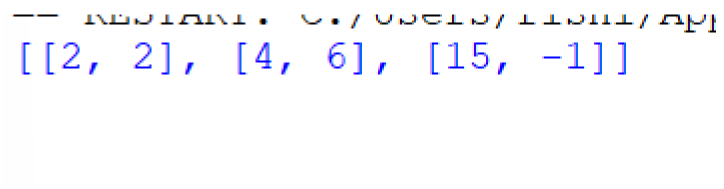
PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)

File Edit Format Run Options Window Help

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def inorder_traversal(root):
    if root is None:
        return []
    return inorder_traversal(root.left) + [root.val] + inorder_traversal(root
def closest_nodes(root, queries):
    sorted_vals = inorder_traversal(root)
    result = []
    for q in queries:
        mini = -1
        maxi = -1
        left, right = 0, len(sorted_vals) - 1

        while left <= right:
            mid = (left + right) // 2
            if sorted_vals[mid] <= q:
                mini = sorted_vals[mid]
                left = mid + 1
            else:
                right = mid - 1

        left, right = 0, len(sorted_vals) - 1

        while left <= right:
            mid = (left + right) // 2
            if sorted_vals[mid] >= q:
                maxi = sorted_vals[mid]
                right = mid - 1
            else:
                left = mid + 1

        result.append([mini, maxi])
    return result
root = TreeNode(6)
root.left = TreeNode(2, TreeNode(1), TreeNode(4))
root.right = TreeNode(13, TreeNode(9), TreeNode(15, TreeNode(14)))
queries = [2, 5, 16]
print(closest_nodes(root, queries))
```

OUTPUT :



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

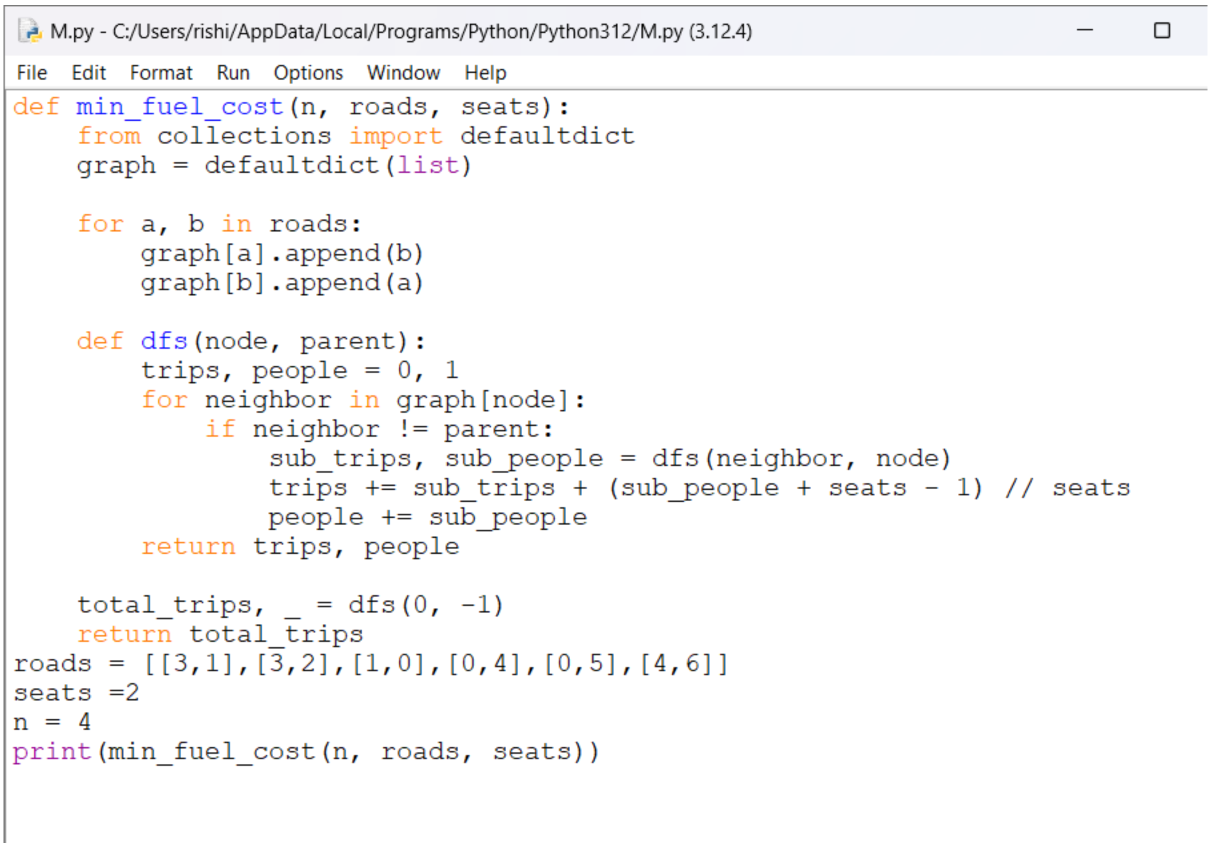
9. Minimum Fuel Cost to Report to the Capital There is a tree (i.e., a connected, undirected graph with no cycles) structure country network consisting of n cities numbered from 0 to n - 1 and exactly n - 1 roads. The capital city is city 0. You are given a 2D integer array roads where roads[i] = [ai, 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

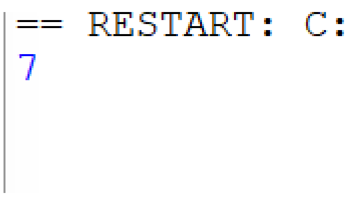
Output: 3 Explanation: - Representative1 goes directly to the capital with 1 liter of fuel. - Representative2 goes directly to the capital with 1 liter of fuel. - Representative3 goes directly to the capital with 1 liter of fuel. It costs 3 liters of fuel at minimum. It can be proven that 3 is the minimum number of liters of fuel

PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)
File Edit Format Run Options Window Help

```python
def min_fuel_cost(n, roads, seats):
    from collections import defaultdict
    graph = defaultdict(list)

    for a, b in roads:
        graph[a].append(b)
        graph[b].append(a)

    def dfs(node, parent):
        trips, people = 0, 1
        for neighbor in graph[node]:
            if neighbor != parent:
                sub_trips, sub_people = dfs(neighbor, node)
                trips += sub_trips + (sub_people + seats - 1) // seats
                people += sub_people
        return trips, people

    total_trips, _ = dfs(0, -1)
    return total_trips
roads = [[3,1],[3,2],[1,0],[0,4],[0,5],[4,6]]
seats =2
n = 4
print(min_fuel_cost(n, roads, seats))
```

OUTPUT :



```
== RESTART: C:
7
```

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:

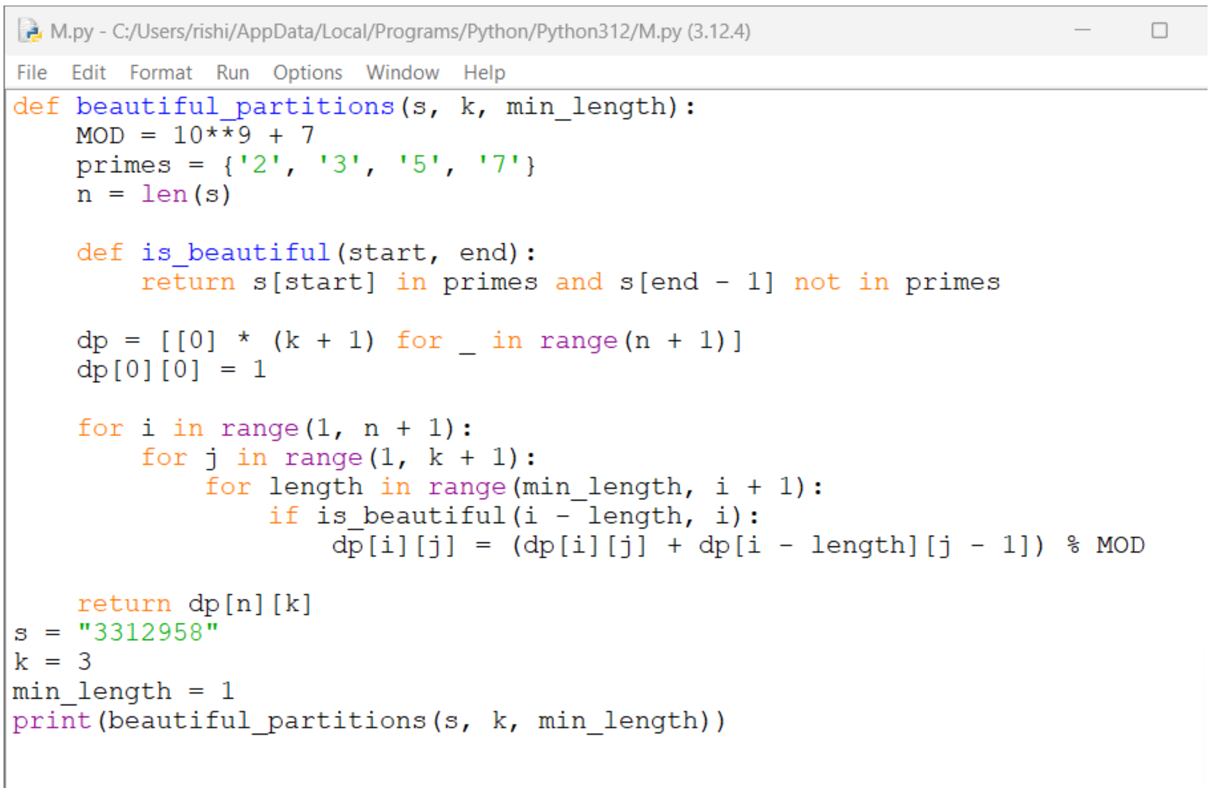
● s is partitioned into k non-intersecting substrings. ● Each substring has a length of at least minLength.

● 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

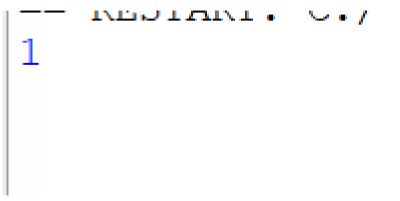
Output: 3 Explanation: There exists three ways to create a beautiful partition: "2354 | 218 | 5131" "2354 | 21851 | 31" "2354218 |.

PROGRAM :


M.py - C:/Users/rishi/AppData/Local/Programs/Python/Python312/M.py (3.12.4)
File Edit Format Run Options Window Help

```
def beautiful_partitions(s, k, min_length):
    MOD = 10**9 + 7
    primes = {'2', '3', '5', '7'}
    n = len(s)

    def is_beautiful(start, end):
        return s[start] in primes and s[end - 1] not in primes

    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 length in range(min_length, i + 1):
                if is_beautiful(i - length, i):
                    dp[i][j] = (dp[i][j] + dp[i - length][j - 1]) % MOD

    return dp[n][k]
s = "3312958"
k = 3
min_length = 1
print(beautiful_partitions(s, k, min_length))
```

OUTPUT :



```
1
```