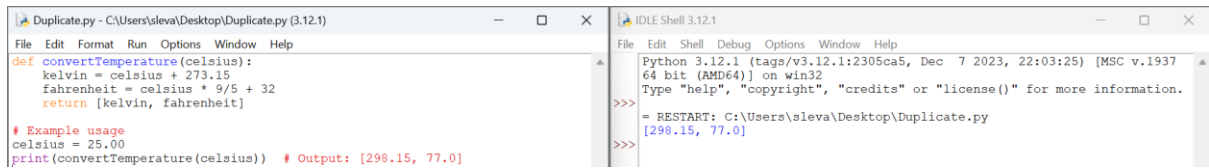


ASSIGNMENT 7

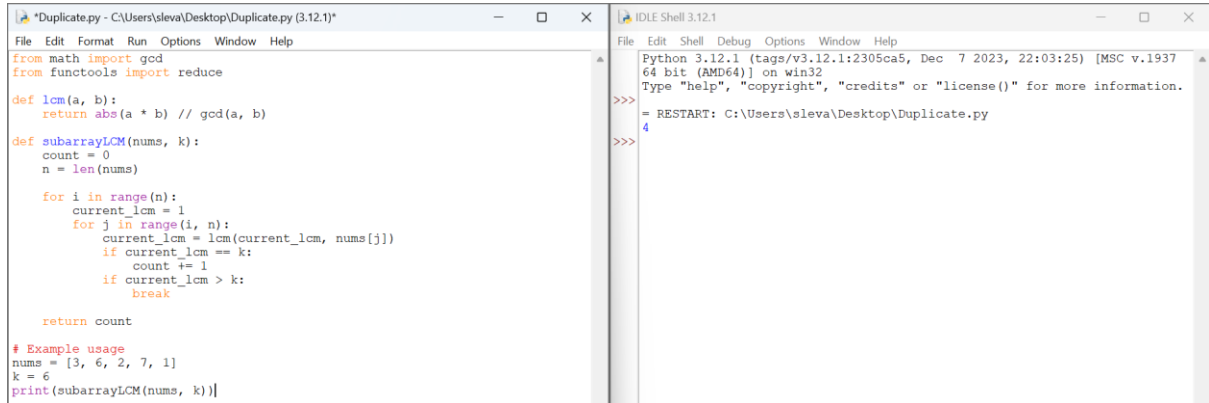
1. Convert to Temperature



The screenshot shows a Python IDE with a file named 'Duplicate.py'. The code defines a function 'convertTemperature' that takes a Celsius value and returns a list with Kelvin and Fahrenheit values. An example usage is provided with celsius = 25.00, resulting in an output of [298.15, 77.0].

```
def convertTemperature(celsius):  
    kelvin = celsius + 273.15  
    fahrenheit = celsius * 9/5 + 32  
    return [kelvin, fahrenheit]  
  
# Example usage  
celsius = 25.00  
print(convertTemperature(celsius)) # Output: [298.15, 77.0]
```

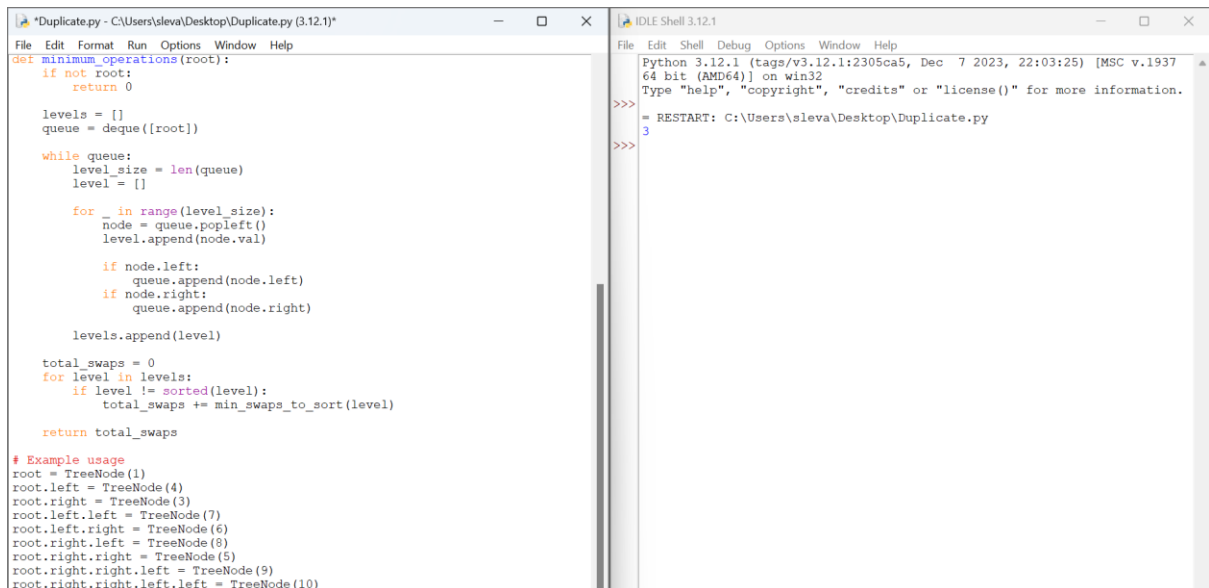
2. Number of Subarrays With LCM Equal to K



The screenshot shows a Python IDE with a file named 'Duplicate.py'. The code defines a function 'subarrayLCM' that counts the number of subarrays with an LCM equal to a given value 'k'. An example usage is provided with nums = [3, 6, 2, 7, 1] and k = 6, resulting in an output of 4.

```
from math import gcd  
from functools import reduce  
  
def lcm(a, b):  
    return abs(a * b) // gcd(a, b)  
  
def subarrayLCM(nums, k):  
    count = 0  
    n = len(nums)  
  
    for i in range(n):  
        current_lcm = 1  
        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  
  
# Example usage  
nums = [3, 6, 2, 7, 1]  
k = 6  
print(subarrayLCM(nums, k))
```

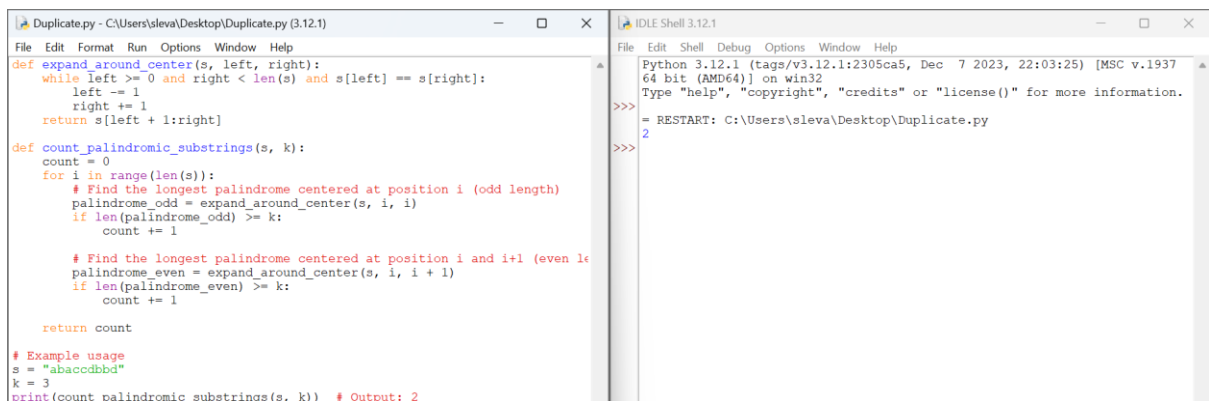
3. Minimum Number of Operations to Sort a Binary Tree by Level



The screenshot shows a Python IDE with a file named 'Duplicate.py'. The code defines a function 'minimum_operations' that calculates the minimum number of swaps required to sort a binary tree by level. An example usage is provided with a specific tree structure, resulting in an output of 3.

```
def minimum_operations(root):  
    if not root:  
        return 0  
  
    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)  
  
    total_swaps = 0  
    for level in levels:  
        if level != sorted(level):  
            total_swaps += min_swaps_to_sort(level)  
  
    return total_swaps  
  
# 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.right.right.left = TreeNode(9)  
root.right.right.left.left = TreeNode(10)
```

4. Maximum Number of Non-overlapping Palindrome Substrings



The screenshot shows a Python IDE with a file named 'Duplicate.py'. The code defines a function 'count_palindromic_substrings' that counts the maximum number of non-overlapping palindromic substrings in a given string. An example usage is provided with s = "abaccdbbd", resulting in an output of 2.

```
def expand_around_center(s, left, right):  
    while left >= 0 and right < len(s) and s[left] == s[right]:  
        left -= 1  
        right += 1  
    return s[left + 1:right]  
  
def count_palindromic_substrings(s, k):  
    count = 0  
  
    for i in range(len(s)):  
        # Find the longest palindrome centered at position i (odd length)  
        palindrome_odd = expand_around_center(s, i, i)  
        if len(palindrome_odd) >= k:  
            count += 1  
  
        # Find the longest palindrome centered at position i and i+1 (even length)  
        palindrome_even = expand_around_center(s, i, i + 1)  
        if len(palindrome_even) >= k:  
            count += 1  
  
    return count  
  
# Example usage  
s = "abaccdbbd"  
k = 3  
print(count_palindromic_substrings(s, k)) # Output: 2
```

5. Minimum Cost to Buy Apples

```
File Edit Format Run Options Window Help
import heapq
def dijkstra(graph, start):
    n = len(graph)
    costs = [float('inf')] * n
    costs[start] = 0
    visited = [False] * n
    pq = [(0, start)]

    while pq:
        cost, node = heapq.heappop(pq)
        if visited[node]:
            continue
        visited[node] = True

        for neighbor, road_cost in graph[node]:
            total_cost = cost + road_cost
            if total_cost < costs[neighbor]:
                costs[neighbor] = total_cost
                heapq.heappush(pq, (total_cost, neighbor))

    return costs
def min_total_cost(n, roads, apple_cost, 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))
    shortest_paths = [dijkstra(graph, i) for i in range(n)]
    answer = []
    for i in range(n):
        min_cost = min(apple_cost[j] + k * shortest_paths[i][j] for j in range(n))
        answer.append(min_cost)

    return answer
n = 4
roads = [[1,2,4],[2,3,2],[2,4,5],[3,4,1],[1,3,4]]
apple_cost = [56,42,102,301]
k = 2
print(min_total_cost(n, roads, apple_cost, k)) # Output: [54, 42, 48, 51]
```

```
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleval\Desktop\Duplicate.py
[50, 42, 46, 48]
>>>
```

7. Number of Unequal Triplets in Array

```
File Edit Format Run Options Window Help
def count_triplets(nums):
    count = 0
    freq = {} # Dictionary to store the frequency of each element

    # Count the frequency of each element in nums
    for num in nums:
        freq[num] = freq.get(num, 0) + 1

    # Iterate through each unique pair of elements in nums
    for i in range(len(nums)):
        for j in range(i + 1, len(nums)):
            # If the pair is distinct and there exists a third element greater
            if nums[i] != nums[j] and nums[i] < nums[j] and freq.get(nums[j]) > 1:
                count += 1

    return count

# Example usage
nums = [4, 4, 2, 4, 3]
print(count_triplets(nums))
```

```
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\sleval\Desktop\Duplicate.py =====
1
>>>
```

8. Closest Nodes Queries in a Binary Search Tree

```
File Edit Format Run Options Window Help
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def inorder(root, query, min_val, max_val):
    if root is None:
        return

    if root.val <= query:
        min_val[0] = root.val
        inorder(root.right, query, min_val, max_val)
    else:
        max_val[0] = root.val
        inorder(root.left, query, min_val, max_val)

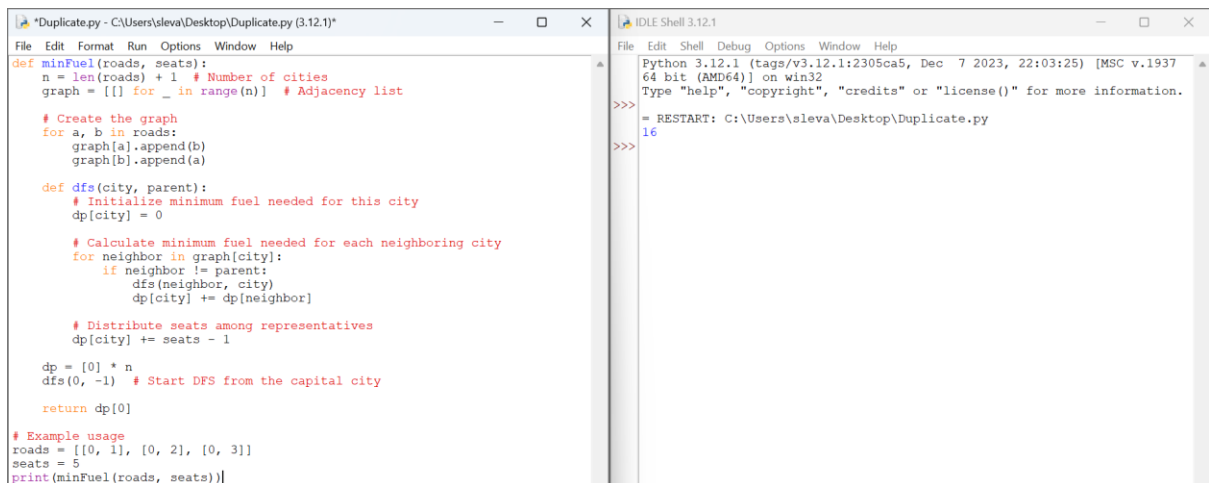
def processQueries(root, queries):
    result = []
    for query in queries:
        min_val = [-1]
        max_val = [-1]
        inorder(root, query, min_val, max_val)
        result.append([min_val[0], max_val[0]])
    return result

# Example usage
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("Output:")
print(processQueries(root, queries)) # Output: [[2, 2], [4, 6], [15, -1]]
```

```
File Edit Shell Debug Options Window Help
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleval\Desktop\Duplicate.py
Output:
[[2, 4], [4, 6], [15, -1]]
>>>
```

9. Minimum Fuel Cost to Report to the Capital



```
*Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)*
File Edit Format Run Options Window Help
def minFuel(roads, seats):
    n = len(roads) + 1 # Number of cities
    graph = [[] for _ in range(n)] # Adjacency list

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

    def dfs(city, parent):
        # Initialize minimum fuel needed for this city
        dp[city] = 0

        # Calculate minimum fuel needed for each neighboring city
        for neighbor in graph[city]:
            if neighbor != parent:
                dfs(neighbor, city)
                dp[city] += dp[neighbor]

        # Distribute seats among representatives
        dp[city] += seats - 1

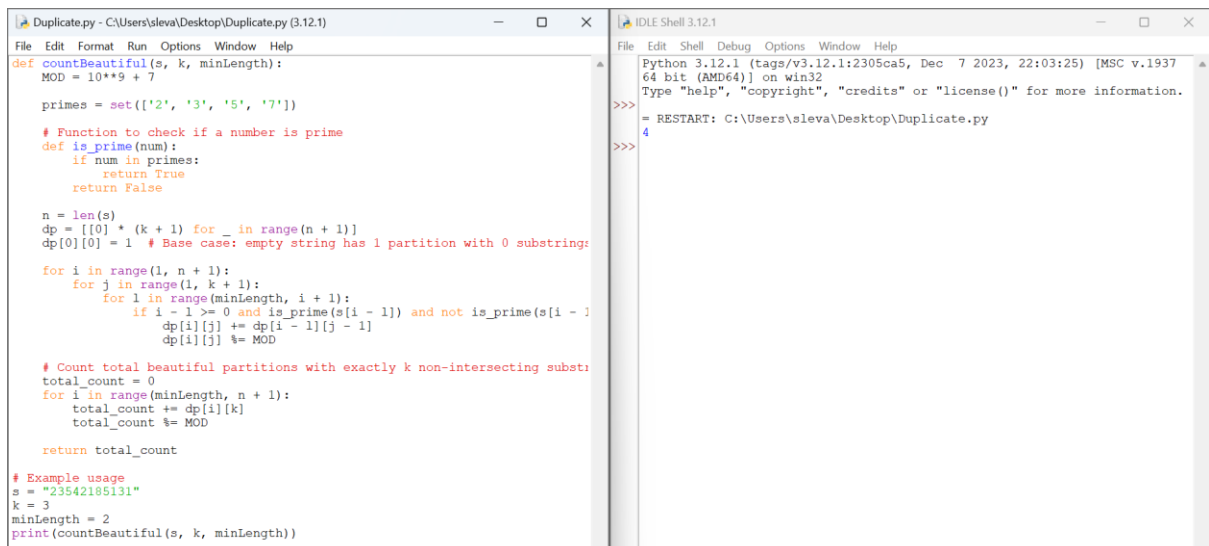
    dp = [0] * n
    dfs(0, -1) # Start DFS from the capital city

    return dp[0]

# Example usage
roads = [[0, 1], [0, 2], [0, 3]]
seats = 5
print(minFuel(roads, seats))

Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleval\Desktop\Duplicate.py
16
>>>
```

10. Number of Beautiful Partitions



```
Duplicate.py - C:\Users\sleval\Desktop\Duplicate.py (3.12.1)
File Edit Format Run Options Window Help
def countBeautiful(s, k, minLength):
    MOD = 10**9 + 7

    primes = set(['2', '3', '5', '7'])

    # Function to check if a number is prime
    def is_prime(num):
        if num in primes:
            return True
        return False

    n = len(s)
    dp = [[0] * (k + 1) for _ in range(n + 1)]
    dp[0][0] = 1 # Base case: empty string has 1 partition with 0 substrings

    for i in range(1, n + 1):
        for j in range(1, k + 1):
            for l in range(minLength, i + 1):
                if i - l >= 0 and is_prime(s[i - l] and not is_prime(s[i - l - 1]):
                    dp[i][j] += dp[i - l][j - 1]
                    dp[i][j] %= MOD

    # Count total beautiful partitions with exactly k non-intersecting substrings
    total_count = 0
    for i in range(minLength, n + 1):
        total_count += dp[i][k]
        total_count %= MOD

    return total_count

# Example usage
s = "23542185131"
k = 3
minLength = 2
print(countBeautiful(s, k, minLength))

Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\sleval\Desktop\Duplicate.py
4
>>>
```